

版权注意事项：

- 1、书籍版权归作者和出版社所有
- 2、本PDF仅限用于个人获取知识，进行私底下的知识交流
- 3、PDF获得者不得在互联网上以任何目的进行传播
- 4、如觉得书籍内容很赞，请购买正版实体书，支持作者
- 5、请于下载PDF后24小时内删除本PDF。



微信小程序首批内测开发者

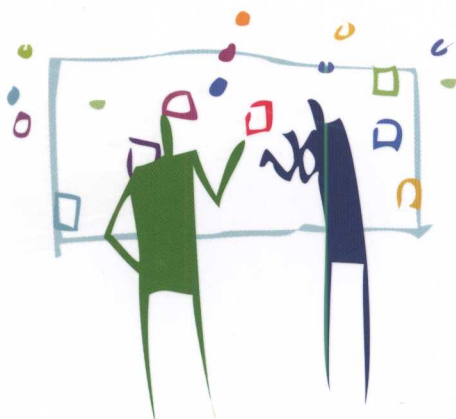
知乎技术类专栏“小楼昨夜又秋风”作者专业奉献

以Orange Can项目为主线，深入浅出地介绍微信小程序的基本结构、开发模式、组件应用、数据绑定方法、微信API使用、微信与设备硬件交互、微信支付等内容

总结小程序开发中踩过的“坑”、常见的开发误区以及开发心得，让你减少试错时间，快速开发出自己的小程序

移动开发丛书

提供本书
小程序项目资源文件
和源代码下载



根据微信小程序
公开上线新版本编写

微信小程序开发 入门与实践

雷磊 编著

清华大学出版社





从童年呼啸而过的火车

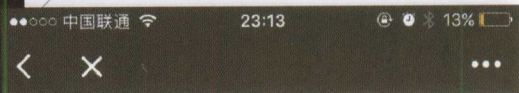
林白衣

24小时前

小时候，家的后面有一条铁路。听说从南方北上的火车都必须经过这条铁路。火车大多在晚上经过，可呜呜的汽笛声，往往却被淹没在傍晚小院儿里散步的人群声中。只有在夜深人静的时候，火车的声音才能清晰的从远处飘过来。虽然日日听见火车的汽笛声，可说也奇怪，我竟从来不知道铁路在哪里。在每个夏日午后，我都会有一种去寻找铁路的冲动，去看看这条铁路究竟是从哪里来，又将通向哪里去

♥ 22 💬 4 ★ 7

文章详情页面真机效果图



评论..... (共4条)



青石

那一年的毕业季，我们挥挥手，来不及说再见，就踏上了远行的火车。



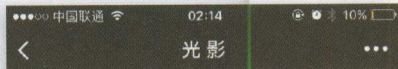
2017-01-18 15:09



林白



8"



正在热映

更多 >



功夫瑜伽

★★★★★ 5.6



东北往事之破...

★★★★★ 0



西游伏妖篇

★★★★★ 5.7

即将上映

更多 >



萤火奇兵

★★★★★ 0



极限特工3: ...

★★★★★ 6



决战食神

★★★★★ 0

豆瓣Top250



文字

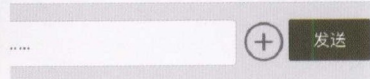


光影



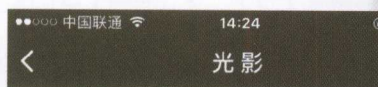
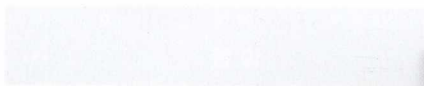
设置

更多 >



评论页面真机效果图

电影首页真机效果图



射雕英雄传



射雕英雄传

★★★★★ 8.1



射雕英雄传

★★★★★ 6.5



射雕英雄

★★★★★



射雕英雄传

★★★★★ 9.1



射雕英雄传

★★★★★ 8.6



射雕英雄传之...

★★★★★ 8.7



文字

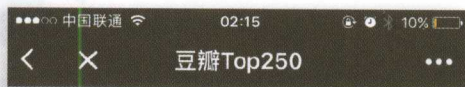


光影



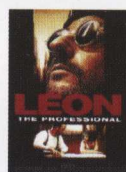
设置

电影搜索真机效果图



肖申克的救赎...

★★★★★ 9.6



这个杀手不太...

★★★★★ 9.4



霸王别姬

★★★★★ 9.5



阿甘正传

★★★★★ 9.4



美丽人生

★★★★★ 9.5



千与千寻

★★★★★ 9.2



辛德勒的名单...



泰坦尼克号



盗梦空间





• 设置页面真机效果图

剧情简介

赛车手阿浪一直对父亲反对自己的赛车事业耿耿于怀，在向父亲证明自己的过程中，阿浪却意外的卷入了一场奇妙的冒险。他在这段经历中结识了一群兄弟好友，一同闯过许多奇幻的经历，也对自己的身世有了更多的了解。

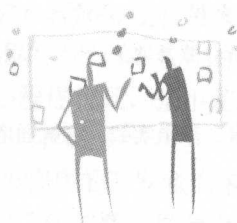
影人



• 电影详情页面真机效果图

· 移动开发丛书 ·

微信小程序开发 入门与实践



雷磊 编著

清华大学出版社
北京

内 容 简 介

本书主要围绕 Orange Can 项目展开一系列编码工作,用几近真实的项目介绍小程序的各个 API、组件用法,并附带一些小程序开发的经验、技巧以及常见的误区说明。整个 Orange Can 项目分为三部分:文章阅读、电影资讯以及设置。文章阅读包括文章列表、文章详情以及评论,通过编写文章阅读功能的代码,读者将学会 swiper 组件的裁剪模式、image 组件的裁剪模式、缓存的使用技巧、列表渲染、数据绑定、模板、音乐播放、录音、分享等知识。除此之外,读者将对小程序页面的生命周期有一个大致了解。学习完这部分内容,读者将可以轻松做出一个内容型小程序应用。电影资讯功能主要介绍如何调用服务器数据及 template 模板的使用技巧。设置页面功能包含大量功能示例,包括获取硬件设备信息、罗盘与重力感应的应用、扫描二维码、用户登录、用户信息校验、解析用户加密数据、获取用户 openId、发送模板消息、微信支付等功能。

本书还提供部分服务器的 PHP 代码,主要供用户登录、校验、解析加密数据、模板消息、微信支付等功能调用。

本书内容丰富、注重实战,讲解通俗易懂。适合小程序开发人员、培训机构和企业内部培训使用。

本书封面贴有清华大学出版社防伪标签,无标签者不得销售。

版权所有,侵权必究。侵权举报电话:010-62782989 13701121933

图书在版编目(CIP)数据

微信小程序开发入门与实践/雷磊编著. —北京:清华大学出版社,2017

(移动开发丛书)

ISBN 978-7-302-46801-1

I. ①微… II. ①雷… III. ①移动终端—应用程序—程序设计 IV. ①TN929.53

中国版本图书馆CIP数据核字(2017)第052715号

责任编辑:王金柱

封面设计:王翔

责任校对:闫秀华

责任印制:何莘

出版发行:清华大学出版社

网 址: <http://www.tup.com.cn>, <http://www.wqbook.com>

地 址:北京清华大学学研大厦A座 邮 编:100084

社总机:010-62770175 邮 购:010-62786544

投稿与读者服务:010-62776969, c-service@tup.tsinghua.edu.cn

质 量 反 馈:010-62772015, zhiliang@tup.tsinghua.edu.cn

印 刷 者:三河市君旺印务有限公司

装 订 者:三河市新茂装订有限公司

经 销:全国新华书店

开 本:190mm×260mm 印 张:23 插 页:2 字 数:589千字

版 次:2017年4月第1版 印 次:2017年4月第1次印刷

印 数:1~3500

定 价:69.00元

产品编号:073496-01

推荐序

什么是小程序？小程序在内测的时候，微信之父张小龙曾写了这样一段文字：“小程序是一种不需要下载、安装即可使用的应用，它实现了触手可及的梦想，用户扫一扫或搜一下就能打开应用，实现了用完即走的理念，用户不用安装太多应用，应用随处可用，但又无须安装卸载”。

有些人说：“轰轰烈烈的小程序其实也就真正火了一天。”2016年，小程序预热，张小龙第一次提前一年高调把小程序拿出来讲（当时还叫应用号），被媒体炒得沸沸扬扬。一年后，上线的当天刷爆朋友圈和微信群。一周以后，当初轰轰烈烈的小程序并没有带来想象中的轰动，很少见到讨论它的内容。上线才4天“罗辑思维”就宣布撤出，“今日头条”的小程序暂停了又恢复，剧情变化得太快，让人看不清小程序的前景，于是“小程序已死，张小龙或走下神坛”的论调随处可见。

但真的是这样的么？我认为还言之过早。第一批尝鲜小程序的开发者大多是互联网App的开发者，已经习惯了从平台获取流量的思路，在日活跃用户7亿的微信平台上推出新产品，大家都想占据先机，获取用户和流量。然而小程序不提供入口，不提供推荐，分享也不能分享到朋友圈。这无疑让早期抱着“蹭”流量目的的小程序开发厂商大失所望。

小程序适合的是场景化、服务型的产品。张小龙在演讲中清楚地描述了他想象中小程序应用的典型场景：在汽车站购买车票的时候扫码购票，无须在售票窗口排队；等公交时扫码启动程序，查询下一辆车到达的时间……

“摩拜”单车的小程序就是一个很好的基于场景化的应用。当你在马路上走累了看到一辆“摩拜”单车时，可直接通过微信的“扫一扫”扫描“摩拜”单车上的二维码，进入摩拜小程序页面，轻松解锁身边的“摩拜”单车开始使用，而不用从App Store里下载一个几十兆的App安装。

在我看来，小程序是张小龙对未来应用程序形态的希望。如今App实在是太多了，正如2009年iPhone广告语所说“there's an app for that”。2016年6月，iOS App已经超过200万个，几乎做到了“there is an app for just about anything”。作为一个智能手机的重度使用者，我的iPhone手机里安装的App超过了400多个，很多App需要通过搜索找到后启动，但是常用的不超过20个，每天都使用的不超过10个。很难想象5年后、10年后的应用程序还是这种形态，最好的状态莫过于当我有需要的时候它出现，解决我的需求之后它消失，大概就是张小龙所谓的“触手可及、用完即走”吧。

小程序不论后面的发展怎么样，都是微信团队对于未来应用形态的一种探索和尝试。难怪张小龙在小程序内测的时候在朋友圈晒了一张图，并配文说“程序猿的一小步，程序的一大步”。作为开发者，要时刻保持学习，保持对技术的敏感、对变革的敏感，只有这样在未来才不会掉队。

雷磊是我认识多年的好朋友，资深技术专家，学习能力极强，时刻对新技术保持敏感，特别善于独立思考、分析、解决问题。他的第一本书《微信小程序开发入门与实践》一定会给想要尝试小程序的你很多收获。更重要的是，本书渗透很多雷磊自己学习的“套路”，遇到一个问题，先用最简单粗暴的方式解决，然后分析这个方案的优缺点，再找出更优秀的解决方案，通过不断“重构”，渐进式地加入对知识的理解。学会这种思维方式之后，在自身技术成长的道路上必将终身受用！

石墨文档联合创始人 陈旭

2017年3月

我眼中的小程序

这本书的编写历史很有意思。从2016年10月中旬开始构思本书的内容，到2017年3月定稿，历时5个月的时间。这中间小程序的大起大落颇值得我们认真回味一下。

2016年9月，小程序内测后迅速引爆了整个互联网圈，含着金钥匙出生的小程序，甚至还没有展现其全貌就被捧上了天。App完了、微信要成操作系统、厉害了我的大Web前端，各种不着调儿的声音甚嚣尘上。不过，作为一个技术开发者，我深刻明白，没有完美的技术，更不会有突然就消亡的产品，小程序不会一统江湖，App也不会销声匿迹，小程序一定是以一种和App互补的姿态出现在人们的视野，至少短期之内一定是这样。

2016年9月到12月28日这段时间，所有人都看不透小程序，连入口、发现机制都不知道，如何能够精准地定位小程序？

直到2016年12月28日微信公开课上，张小龙终于用一连串“没有”告诉了所有媒体和开发者，你们都是错的。小程序并不是你们想的那样：我说它只是用完即走的服务，那它就一定是用完即走的服务，不会带有很强的媒体属性；我说没有应用市场，那官方就没有应用市场；我说扫一扫、搜一下就能打开小程序，那就没有第三种途径可以触及小程序。在当时，这些“没有”确实让很多开发者大吃一惊。

官方没有应用市场，只支持有限能力的搜索（目前，微信已开放了小程序的模糊搜索），不能分享到朋友圈，不能被关注，没有主动推送消息的能力，这是官方在2016年末时向开发者所描绘的小程序肖像。很多产品经理、创业者在看完微信公开课后，都默默地在想，那我要小程序干什么？

在2016年12月28日的微信公开课上，张小龙还公布了小程序正式开放上线的日期：2017年1月9日。10年前的1月9日，乔布斯公布了第一代iPhone，这个产品重新定义了智能手机；10年后的1月9日，微信发布了小程序，这个产品能重新定义应用程序吗？

诸多限制并没有让开发者特别沮丧，无数创业者依然坚信小程序是一个巨大的红利，不试试怎么知道呢？

2017年1月9日凌晨整点刚过，积压了一年多（早在2016年1月，张小龙就公布了微信应用号）热情的开发者迅速上线了自己的小程序，无数早已审核完毕的小程序如潮水般涌来。可创业者失望了，原来小程序并不如大家想得那般美好，信守承诺的微信不仅没有放开任何一个限制，反而屏蔽了长按二维码识别小程序的功能（目前已开放）。长按二维码是当时可以很方便地在线上推广小程序的途径，可这条路也被微信无情地斩断了。

顿时互联网圈子不淡定了，许多人在前两个月对小程序的看法还是“万紫千红总是春”，但转眼间却发现小程序并不像想得那样美好。线上的推广渠道几乎没有、搜索必须输入完整的名字（目前已解除这个限制，支持模糊搜索）、不能和公众号同名、用户只能从历史中打开曾经使用过的小程序、不对个人开发者开放，这些都是小程序的真实“模样”。所谓“人生若只如初见，何事秋风悲画扇”大抵就是这样的心情。很多创业者、开发者不再认可小程序的模式，整个互联网对于小程序的评价来了一个180度的急转弯。创业者认为小程序不利于推广，开发者认为这就是一个Web网页，并没有什么新意，很多普通用户甚至根本不知道有小程序这个东西。

完全摒弃线上入口，真的好吗？开放模糊搜索、允许公众号关联小程序，这明明在技术上并不难实现，为何要如此固执。也许是微信已经习惯了低调与克制，只是这次的“克制”却显得如此的生硬。当然，也许微信有自己的考虑与布局，只是我们局外人不得而知。

2017年3月27日，画风突变，微信公布了小程序的若干新特性：对个人开发者开放注册；公众号自定义菜单与模板消息可以绑定小程序，用户点击后可直达小程序；移动App可分享小程序；不再需要扫描小程序特定的二维码，直接扫描商户原有普通二维码也可以直接打开小程序。小程序开始尝试借助微信现有资源来推动小程序的发展。

很明显，小程序的入口逐渐丰富了起来。借助微信公众号那惊人的体量与访问量，被开发者一直诟病的没有传播途径这个问题，也得到了较好的解决。在没有放开这些能力之前，我曾经一度认为小程序和微信并没有什么联系，它完全是一个游离于微信体系之外的产品，和微信的关联感太弱了。但这次新能力的放出，让我感觉到了小程序真的变成了微信“的”小程序。

现在的小程序已初步具备了线上线下同步的雏形。微信绕了一大圈儿，最终还是回归到了线上线下结合这个互联网运营的经典套路。此次推出新特性的意图也较为明显：线下开疆拓土，线上巩固固有优势和辅助线下发展。至于开发者到底是重线上还是重线下，或是二者结合，这就在于开发者的抉择了。反正线上线下都给了开发者，那就自己看着办吧。

但以现在线上产品的完备程度来看，留给新产品的空间已经很小了。尝试着将目光投向线下也许是一个不错的选择，转变一下思维也许会有片新天地。

以前我们的思维是线下什么不方便，那就把线下移植到线上。比如交水电费不方便，那就线上交；火车飞机出行购票不方便，那就线上买。这些服务毫无疑问是互联网发展的必然衍生物。可还有很多活动是线上做不了的，或者说很多线下活动如果放在线上，就会变得索然无味。比如看电影，如果线上能代替线下，为什么还有那么多人购买电影票去看大荧幕？因为电影是一门光与影结合的艺术，需要环境和氛围的辅助才能体会光、影、音结合的美妙和震撼；再比如朋友聚会、同学聚会，大家在线上用微信视频一起聊个天行吗？这自然不行，因为感觉不到位。我们无法脱离线下，却又被线下的不方便所困扰。

在我看来，线上线下结合是一个必然趋势，微信希望的也是从线下挖掘新的流量与场景。线上的服务虽然很方便，但场景有限，人不可能纯粹地活在线上环境里，我们依然有大量线下活动。目前，互联网所能提供的服务品质和服务种类其实已经达到了一个极限，在科技没有革命性进步的情况下，还想从线上有所突破是不太现实的。看似AI、物联网、VR很火热，但它们离真正的



普及还有很长一段路要走。当然，哪天人类的科技真能发展到像电影《星际穿越》中所描绘的智能、多维的层级，这就另当别论了。目前来看，线下的活动场景和机会还是挺多的，只不过还没有被充分挖掘。

所以，不要太过于执着于线上，换一种思路，挖掘一下线下的场景，也许是创业的一个突破口。线下有太多小程序非常适合的场景，拥有巨大的想象空间。虽然现在有不少 H5 服务于线下场景（如很多餐饮业提供的排号及点餐服务），但无论是从体验还是产品闭环上，都远远不及小程序。这点希望开发者能够去思考一下，不要强行用固化的思维看待小程序。这和技术无关，只和场景有关。

那么原生 App、H5 和小程序该如何抉择？至少从目前来看，如果只是一个纯粹的单一的线上小程序，相比于原生 App 还是有一定的劣势的，但在微信将小程序与公众号做了“强绑定”后，小程序的应用又变得非常的多元化，它们之间不再是“非此即彼”了；而在对比 H5 页面时，小程序的体验（使用体验与服务体验）还是要略有优势的，毕竟小程序是一个体系，但缺点是没有 H5 开放和自由，H5 依然是最佳的粘合剂，很适合嵌入到各个种类的产品与应用中。

无论创业者选择 App 还是小程序，或者二者齐头并进，都应当把做好产品放在第一位。不是说选择不重要，而是过分在意选择载体而忽略了本质，多少有些舍本逐末。回归产品本质，解决用户的问题也许才是每个开发者应该斟酌和思量的事情。

对于小程序，更多时候我只是一个看客，它发展得好与不好对我而言没有太大影响。它发展得好，我这本书就有一定价值和意义，它发展得不好，未来这本书就如废纸一般，被读者丢弃在角落，蒙上一层厚厚的灰尘。我只是一个普通的开发者，这本书的内容也主要是讲解和技术本身相关的知识。它不是一本运营手册，仅仅是一本技术书籍，无论小程序未来发展如何，读者都可以从这本书里获取一些有用的知识。平台是多变的，但技术思维是相通的。

抛开繁杂的因素，仅从开发体验上讲，我非常喜欢小程序的灵巧。下载源代码后，不用配环境，只需要一个开发工具就能跑起小程序，这在现在越来越复杂的开发平台中是不多见的。也许小程序以后专业用来做原型代替 Axure 也不是不可能。此外，小程序非常适合开发者去践行自己的想法，以前移动端的开发成本太高，有很多想法因此都无法实现。但现在有了小程序就不同了，完全可以快速编写一个 MVP 产品，然后迅速投向市场去试错，再逐步迭代、完善，最终考虑是否要推出自己的原生 App。

也许未来真的像张小龙所言，处处都是二维码；也可能未来的小程序就像历史上无数被技术更迭的浪潮所淹没的语言和平台一样，遁于无形。可谁又能预测未来呢？互联网每天都有新贵诞生，也有王者陨落，成与败都在弹指之间，这在互联网的世界里是再平常不过的事情。对于一个寄生于超级 App 的平台而言更是如此，一切都是未知数。

但无论如何，小程序这种即用即走的理念确实是现今主流 App 所缺失的。世界和科技总在不断变迁和进步，小程序这种形态的应用能否打破现在移动端的格局，我们目前还不得而知，但我相信这种理念确实值得认真思考一下，至于移动端的未来，还是交给时间吧。

最后，我花了很多时间思考这本书应该起个什么名字。从入门到精通？从来没有一本书能让你真的精通一项技术，精通一项技术是需要经年累月的持续学习的；零基础学习小程序？这也不太实际，本书不是讲解 JavaScript 等基础语言，很难做到零基础学习；从入门到实战？好像挺符合本书的定位。但我思忖，实战这个词太噱头，哪有在书里就能实战的，书里谈的代码从来都无法模拟真实的运行环境，何能称为实战？我诚惶诚恐，不敢用“实战”二字，思来想去，最后还是用“实践”落笔书名。以实例践行小程序的应用，同读者一起探讨小程序的开发。

雷 磊

2017年2月初稿

3月修订

前言

本书的特点与特色

兴许是我向来不喜欢很多编程书籍开篇就大篇幅罗列知识点的做法，从业八年以来，每每翻阅技术类书籍，看到连篇累牍的概念理论就头疼不已。接到清华大学出版社的邀约后，我长久思忖如何组织小程序开发这本书的编写思路，写出一本我自己也喜欢看的书籍。

如果能让读者身临其境地开发一个几近真实的项目，在不知不觉中就可以学会小程序开发，那该多好。庆幸的是，小程序不是一门语言，它不需要像 Java、Python、JavaScript 等基础语言教学一样罗列一个个基础语法，它最好的学习方式就是本书的“实践式”学习。因此，本书将用一个较为完整的“案例项目”把小程序的各个知识点“串接”起来，一边做项目，一边学习小程序的开发。做完一个项目就可以入门小程序是本书的目的。

我喜欢这种“实践式”学习所带来的“代入感”（如果你玩过各类角色扮演游戏，你就明白什么是代入感），跟着本书一步步 coding，你不仅收获了知识，更是直接完成了一个像模像样的小程序，这种成就是学习编程最大的动力。即使你是一个基础较好的开发者，只看官方的文档也能学会小程序的开发，我依然建议你认真阅读本书，因为本书将为你节约大量“试错”时间。

本书在很多时候并没有直接给出一个问题的最优解决方案，而是首先给出一个看似很蠢的思路来解决问题。因为这是我们最直接的思维，也是最简单的解决方案。通过分析这个解决方案有什么缺点，最后给出一个更加优秀的解决问题的建议。我想，这符合我们编程里“重构”的概念。相比于直接给出最优解（事实上编程里很难有最优解，只是相对“优秀”），渐进式的解决问题更加能让读者体会到优秀解决方案的优势，避免对知识的生搬硬套。

我一直认为，本书的编写思路也是程序员自学的思路，由点及面、由具体到抽象。在工作中遇到了问题，想办法解决问题，查阅资料学习这个问题的相关知识点，最后把这些知识点总结、归纳，形成自己的知识体系，这是一个通用的学习“套路”。编程的各类语言、框架太多了，技术发展的速度也快得惊人，即使类似功能的框架也多达十几个，我们很难像学习经典数学、基础物理学、现代经济学这样先学习理论再付诸实践。Coder 有时就要有这种直面未知的勇气：先解决、再学习，管它三七二十一。

本书虽然定位于入门，但其中不乏一些小程序的进阶知识，这主要体现在微信开放接口上。学习微信开放接口不仅需要你拥有前端的知识，更要有一定的服务器编程经验，否则你很难理解为什么微信要这么设计开放接口的调用流程？为什么需要这么复杂的签名与令牌体系？

退一步讲，不理解也没关系，遇到类似问题和功能时，你知道怎么去解决即可。在编程里，我们不理解的东西太多了，谁能保证我们将做过的项目、产品每一个细节都理解得清清楚楚、明明白白？有时候记住怎么去做，比为什么这么做更加重要。理解清楚只是一个相对的概念，没人能够准确定义理解到什么程度才能称为“理解清楚”，也没有人能够说明深入到什么程度才算是“深入学习”。所以，有选择地学习原理，把更多精力放在解决问题上，我认为是一个正确的选择。

在本书中，当遇到你不熟悉的知识体系时，没有关系，先写上去，实现这个功能。当以后有了更丰富的经验再回过头来看看这些知识点即可。本书的详细程度完全可以让你即使不懂某个知识点，也可以完成整个 Orange Can 项目。

小程序开发需要的前置技能

如果不考虑服务器，小程序开发只需要开发者具有 JavaScript 和 CSS 相关知识即可。

有很多文章说，开发者开发小程序还需要掌握 Vue、AngularJS，这有些强人所难了。小程序确实有很多和 Vue、AngularJS 相似的地方，这主要体现在数据绑定上。但 Vue 和 AngularJS 远比微信小程序要复杂得多，为了开发一个简单的小程序，学习远比这个简单的东西复杂多倍的框架实在没有必要。

从先来后到的角度看，Vue、AngularJS 等经典 MVVM 框架确实先于小程序出现，且小程序借鉴了许多这些 MVVM 框架中的经典思想。但对于既没有开发过小程序，也没有任何 AngularJS、Vue 经验的开发者，这个先来后到的理念对你没有任何意义。反正都不会，自然是优先学习简单的，再进阶复杂的框架。如果你是一个 iOS 和 Android 转型过来的开发者，完全没有必要理会 Vue 和 AngularJS，小程序开发中的很多思想相信你在自己的 iOS 和 Android 领域已有体会。

如果你只是为了开发小程序前端部分，更没有必要学习 NodeJS。前端是前端，服务器是服务器，我们能把一端做到极致就已经非常了不起了。如果你想一个人开发一个完整的小程序，那服务器语言也没有规定必须是 NodeJS。选择一个你喜欢的服务器语言，PHP、NodeJS、Python、Java、C#、Ruby 都是可以的。

小程序开发需要掌握什么，在我看来是一个伪命题。小程序应该成为零基础入门开发者学习前端的首选开发平台（以学习与实践为目的），因为它足够简单，又同现在的主流 MVVM 框架非常类似，学习曲线很平滑。它应该成为入门其他更复杂、功能更强大的框架的“垫脚石”。

对于一个传统的 Web 开发者，在编写小程序时只需要注意以下两点：

- (1) 小程序中没有 DOM，请放弃“首先获取 DOM，再操作 DOM”的思维。
- (2) 替代 DOM 操作的方法是“数据绑定”。控制组件显示隐藏、切换 CSS 样式、控制滚动条，这些很容易用 DOM 思维思考常见功能在小程序中都是通过“数据绑定”实现的。

如果你想将代码写得更加优美和简洁，那么补充一些 ES6 和 LESS 的知识就更好了。

小程序开发难吗

说小程序是所有开发框架/平台里最简单的可能略微有些夸张，但说小程序是目前所有主流移动开发技术中最简单的毫不夸张。这种简单来自于两个方面：

第一，编写小程序只需要掌握 JavaScript 和 CSS 两门语言。前端最难的是有太多 Web 前端框架、类库需要学习。但是，小程序里的 JavaScript 是“裸奔”的，我们在 Web 开发中常用的各类框架/类库在小程序中统统无法使用。jQuery、Zepto、AngularJS、HightCharts、ECharts，这些 Web 前端学习中的一座座大山，小程序已经全部“干掉了”——小程序运行在一个 JSCore 中，它本身不支持 Web 中的 window 及 DOM 对象。有些 JavaScript 库还是可以使用的，但真的没有必要了，小程序已经提供了简单的架构和内置的特性避免使用这些框架。例如，小程序默认使用



babel 将开发者代码所使用的 ES6 语法转换成三端都能很好支持的 ES5 代码，帮助开发者解决环境不同所带来的开发问题。你所需要掌握的是 JavaScript 和 CSS，原则上讲，不再需要学习各类框架和库了。这无疑减轻了很多初学者的负担。

第二，小程序本身就是为轻量级应用所设计的平台，无论是开发工具、设计规范、API 设计，无不散发出一种“大道至简”的气息。你只需要从官方下载一个开发工具即可立即开始开发小程序，没有复杂的安装环境，没有复杂的目录结构，也没有复杂的打包、部署流程。小程序很多近似死板的规范无疑让开发者减少了很多工作量（不给你选择，自然简单）。关于对于开发者的约束，这个仁者见仁、智者见智，有人喜欢它的简单，也有人憎恨它的“不自由”。

我甚至认为小程序完全可以成为那些完全没有编程经验又想进入前端开发领域的人最好的入门平台。完全可以从小程序入手，通过小程序平滑的学习曲线，在熟悉编程逻辑后，再反向学习 Web 前端庞大的知识体系，从而进入这个行业。

处于早期阶段的小程序

小程序从 2016 年 9 月 22 日公布后，在短短的 3 个月中连续更新了 8 个版本。更新速度非常快。但在 2017 年 1 月 9 日正式开放后，版本迭代速度逐步放慢下来。预计正式上线后，小程序的接口会逐步趋于稳定。

小程序目前还处于极为早期的发展阶段，但其主要的框架、API 都已成型，并不影响我们学习。本书无法保证书籍出版前所使用的 API 不会在出版后有所调整，这个是我不能控制的。本书中所描述的知识与小程序的行为都只在 130400 版本中测试和验证，无法保证以后小程序会不会更改这些行为。事实上，从最开始的内测版本到现在的 130400 版本，小程序不仅调整 API，甚至经常会更改一些 MINA 框架的运行机制和行为，这将导致原本可以正常运行的代码突然变得有问题，需要重新调整。

除此之外，本书的 Orange Can 项目是一个接近于真实项目的小程序，还是有一定复杂度的，我无法保证不出现“任何”bug。事实上，也没有任何人能保证自己的项目不出现“任何”bug，我们能做的就是严格测试、尽量减少 bug 的数量，并在发现 bug 后第一时间做出修正。

虽然无法保证书籍中的代码和 API 完全不会变更，但我会通过微信公众号、知乎专栏等方式同读者交流，保持对书籍的网络更新。同时，也会持续维护 Orange Can 示例项目的源代码，保证项目能够正确、稳定的运行。

截止本书出版之前，小程序最新版本为 0.14.140900。从 130400 到 140900 的版本更迭中，没有再出现因版本更新而造成的问题，项目代码运行良好。可以看到，小程序从 1 月 9 号正式开放后，API 日趋成熟，基本能够保持稳定。

本书与官方文档的区别

本书并不想成为官方开发文档的“搬运工”，除了一些非常必要的地方会引用官方文档的说明，本书几乎不会大段复制官方文档。本书更多使用官方 API 文档完成一个项目，对官方文档的主要内容做详细的补充说明，并附带对文档内容的经验性总结。



官方的 API 文档通常定位于工具类的速查手册而非教程。当然对于开发功底深厚的开发者，只靠开发文档也可以完成一个小程序项目，但通常需要耗费较长时间。开发文档只会告诉开发者有哪些能力，但这些能力如何使用还需要开发者自己探索。

本书可以告诉开发者如何使用官方 API 完成我们自己的业务逻辑，并在这个过程中逐步熟悉官方 API，从而达到入门小程序并自己开发一个小程序的目的。

除此之外，目前来说，小程序的开发还是有不少“坑”的。本书尝试为开发者提前把这些“坑”踩一踩，填实了，以帮助开发者集中精力开发业务，减少浪费在小程序 bug 上的时间。

一本书是永远不可能替代官方 API 文档的。即使想成为官方文档的替代品也是不可能的，因为最新最全的资料永远都由官方文档率先公布，书籍总会有一定的滞后性，这也是我为什么没有在本书中大量引用官方文档的原因。本书的价值在于让开发者快速入门小程序，并讲解小程序的各个特性，降低开发者的学习成本，快速入门开发自己的应用。

官方文档总体来说还是非常全面优秀的，但也有很多讲得不清楚的地方，对于官方文档中错误、遗漏或者没有讲清楚的地方，本书会做详细的补充说明。建议开发者将本书与官方文档结合起来阅读，学习阶段以本书为主，文档为补充；而在开发阶段以文档为主，本书的补充说明为辅。

此外，官方为所有开发者准备了一个非常详尽的 demo，可在微信中搜索“小程序示例”这几个关键字。官方示例 demo 源代码下载地址：

<https://mp.weixin.qq.com/debug/wxadoc/dev/demo.html>

下载项目资源文件、源代码

一本书难以详尽讲解小程序的所有知识点，也无法回答开发中的所有问题，更加难以应对不断更新的小程序版本。这是一个互联网的时代，我们将尝试用书籍+网络的方式来维护本书，并对小程序最新的变更做出代码上的改动，以保证我们的示例代码可以正常运行。读者朋友也可以通过我的微信公众号向我提供反馈，并收到书籍、源代码变更的更新消息。此外，我的知乎专栏也会经常发布一些关于编程和互联网方面的文章。

所有项目源代码、资源文件等内容都将在微信公众号中提供下载地址。读者可以从我的微信公众号中获取项目效果图和最新版源代码。代码包括 Orange Can 项目的小程序代码及部分功能的服务端 PHP 代码。

- 微信公众号：小楼昨夜又秋风
- 知乎专栏：小楼昨夜又秋风
- 知乎 ID：七月在夏天

读者可以访问地址：<http://pan.baidu.com/s/1cxQXie>（注意区分数字和英文字母大小写）获得本书源代码。如果在下载过程中遇到问题，请发电子邮件至 booksaga@126.com，邮件主题设置为“微信小程序开发入门与实践配书资源”。

如何阅读本书

如果你是拥有多年丰富开发经验的开发者，并且已经对小程序有一定了解，建议快速浏览章节目录，找到你感兴趣的主题，然后只看这一小节。



如果你有一定的前端开发基础，但没有小程序的开发经验，建议从头开始做完 Orange Can 项目，深入理解书中的每一个开发技巧。建议先看每个页面的效果图，或者下载最新源代码，在熟悉功能后，自己尝试编写项目功能，再对比源代码的实现方式。也许你写的项目代码比本书中的还要优秀。

如果你想用小程序来入门前端，那么请先熟悉 JavaScript 和 CSS 的基础知识，然后把 Orange Can 项目当作一个真实项目，一边做一边实践所学习的 JavaScript 和 CSS 知识。遇到不太明白的地方努力搞懂它，实在不懂也没关系，先写上去，等有一定经验后再回过头来看。但无论如何，一定要完成 Orange Can 项目的文章和电影这两个部分。

如果你是一个纯粹的技术开发者，那么请直接从第 2 章开始阅读。

总体的建议是，对于 Orange Can 项目的文章和电影部分，应该一步步跟着书籍逐步实现这两个核心功能；而对于“设置”页面，结合源代码“看懂”本书中的内容即可。当然，如果整本书你都能亲手敲打每一行代码，我相信你收获的绝对不仅仅是小程序开发的知识点。

此外，对于 Orange Can 项目的 CSS 样式，建议开发者不要完全照搬本书的 CSS 样式，每个人编写 CSS 样式的思路千差万别，本书无法保证所有 CSS 样式都是有“意义的”，少部分 CSS 样式是为了“防御性”而编写的。重点是小程序相关的知识点和 JavaScript 代码。

微信官方开发者社区

微信官方开放了一个开发者社区，开发者可以在社区里向微信官方反馈 bug 及提出问题。开发者社区地址：

<https://developers.weixin.qq.com/home>

致谢

向我多年的朋友蒋建明致谢，感谢他为本书提供了很多宝贵意见。

感谢石墨文档联合创始人陈旭为本书作序。

感谢清华大学出版社王金柱老师的支持，让我“拖拖拉拉”写了近三个月才完成本书。

雷 磊

2017年2月8日

目 录

第 1 章 微信小程序简介.....	1
1.1 什么是微信小程序.....	2
1.2 什么类型的应用适合用小程序开发.....	5
1.3 小程序与原生 App (iOS、Android) 的优劣对比.....	6
1.4 小程序会淘汰原生 App 吗.....	10
1.5 Web 前端的未来.....	10
1.6 Web 前端开发者与小程序.....	11
1.7 MINA 框架与微信小程序.....	12
1.8 微信小程序 beta 测试版.....	12
第 2 章 小程序环境搭建与开发工具介绍.....	13
2.1 微信 Web 开发者工具下载及安装.....	14
2.2 新建第一个项目.....	14
2.3 微信 Web 开发者工具界面功能介绍.....	16
2.3.1 编辑选项卡.....	17
2.3.2 调试选项卡.....	19
2.3.3 项目选项卡.....	22
2.3.4 编译选项.....	23
2.3.5 后台选项.....	24
2.3.6 缓存选项.....	24
2.3.7 关闭选项.....	24
2.3.8 快速打开官方 API 文档.....	24
2.3.9 开发工具的更新.....	24
2.3.10 常用小程序快捷键.....	25
第 3 章 从第一个简单的“Welcome”页面开始小程序之旅.....	26
3.1 认识小程序的基本文件结构.....	27
3.2 开始动手编写第一个小程序页面.....	28
3.3 构建 welcome 页面的元素和样式.....	31
3.4 小程序所支持的 CSS 选择器.....	35

3.5 Flex 布局	36
3.6 小程序自适应单位 rpx 简介	39
3.7 全局样式文件 app.wxss	42
3.8 页面的根元素 page	42
3.9 app.json 中的 window 配置项	44
第 4 章 文章列表页面	47
4.1 文章列表页面元素分析及准备工作	48
4.2 swiper 组件	50
4.3 Boolean 值的陷阱	53
4.4 构建文章列表的骨架和样式	54
4.5 image 组件的 4 种缩放模式与 9 种裁剪模式	57
4.5.1 scaleToFill	58
4.5.2 aspectFit	58
4.5.3 aspectFill	59
4.5.4 widthFix	60
4.5.5 9 种裁剪模式	60
4.6 完成静态文章列表	61
4.7 .js 文件的代码结构与 Page 页面的生命周期	64
4.8 数据绑定	68
4.8.1 初始化数据绑定	69
4.8.2 在哪里可以查看数据绑定对象	70
4.8.3 绑定复杂对象	71
4.8.4 数据绑定更新	72
4.9 列表渲染 wx:for	76
4.10 配置单个页面导航栏背景色	79
4.11 从欢迎页面跳转到文章页面	80
4.11.1 事件	80
4.11.2 redirectTo 与 navigateTo	82
4.11.3 小程序最多只能有 5 层页面	83
4.11.4 冒泡事件与非冒泡事件	84
第 5 章 模块、模板与缓存	85
5.1 将文章数据从业务中分离	86
5.2 小程序的模块	87
5.3 小程序的模板化	89
5.4 消除 template 模板对外部变量名的依赖	90



5.5	include 与 import 引用模板的区别	92
5.6	CSS 的模块化	93
5.7	令人遗憾的模板化而非组件化	94
5.8	使用缓存在本地模拟服务器数据库	95
5.8.1	应用程序的生命周期	95
5.8.2	使用 Storage 缓存初始化本地数据库	96
5.8.3	缓存的强制清理及注意事项	99
5.9	编写缓存数据库操作类	99
5.10	使用缓存数据库操作类	101
5.11	使用 ES6 改写缓存操作类	102
5.12	完善文章数据	103
5.13	完整的数据.js 数据	104
第 6 章	文章详情页面	110
6.1	跳转到文章详情页面	111
6.2	不要在 template 上注册事件	112
6.3	页面间传递参数的 3 种方式	113
6.3.1	组件的自定义属性	113
6.3.2	通过 dataset 获取组件自定义属性	114
6.3.3	获取页面参数值	115
6.4	编译时设置初始化页面及参数	115
6.5	读取文章详情数据	116
6.6	文章 id 号的数据流向图	117
6.7	编写文章详情页面	118
6.8	垂直居中问题的经典解决方法	121
6.9	动态设置导航栏标题	122
6.9.1	使用配置文件配置导航栏标题	122
6.9.2	使用 wx.setNavigationBarTitle(OBJECT)设置导航条	123
第 7 章	收藏、评论、点赞与计数功能	124
7.1	收藏、评论、点赞、计数功能准备工作	125
7.2	文章收藏功能	127
7.2.1	条件渲染: wx:if 与 wx:else	127
7.2.2	实现收藏点击功能	128
7.2.3	交互反馈 wx.showToast	130
7.3	文章点赞功能	131
7.4	本地缓存的重要性及应用举例	133



7.5	支持文字、图片、拍照、语音上传的文章评论	134
7.6	文章评论页面的实现步骤与思路	134
7.7	获取并绑定文章评论数据	135
7.8	显示文章评论数据	140
7.9	实现图片预览	145
7.10	实现提交评论的界面	146
7.11	wx:if 与 hidden 控制元素显示和隐藏	152
7.12	实现文字评论框和语音评论框的切换	152
7.13	input 组件	153
7.14	bindinput 事件	154
7.15	屏蔽评论关键字	155
7.16	实现自定义发送按钮	157
7.17	同时支持模拟器回车、真机点击“完成”发送评论	161
7.18	图片与拍照评论的界面实现	161
7.19	实现从相册选择照片与拍照	164
7.20	icon 图片	166
7.21	删除已选择的图片	167
7.22	在小程序中使用 CSS 3 动画	168
7.23	实现图片评论的发送	170
7.24	实现语音消息的发送	171
7.25	实现语音消息的暂停与播放	174
7.26	用户授权	176
7.27	解决真机运行时评论页面滑动卡顿的问题	177
7.28	文章阅读计数功能	177
第 8 章	背景音乐播放	180
8.1	显示音乐播放图标	181
8.2	切换音乐播放图标	182
8.3	背景音乐播放的特点	182
8.4	实现单页面背景音乐播放	183
8.5	监听音乐播放	185
8.6	全局变量与全局音乐播放	186
8.7	音乐总控开关	192
8.8	显示音乐的封面图片	194
第 9 章	丰富文章页面	195
9.1	将页面分享给朋友和微信群	196



9.2 从 swiper 组件跳转到文章详情页面	197
9.3 使用小程序动画实现点赞特效	199
第 10 章 电影	204
10.1 小程序的 tab 选项卡	205
10.2 电影页面介绍	208
10.3 编写豆瓣星星评分组件: stars-tpl 模板	210
10.4 编写 movie-tpl 模板	212
10.5 编写 movie-list-tpl 模板	213
10.6 电影首页的骨架与样式	215
10.7 豆瓣电影 API 分析	216
10.8 电影首页的 js 编写	217
10.9 wx.request 发送 http/https 请求	219
10.10 设置 wx.request 的超时时间	221
10.11 处理返回的电影数据	221
10.12 绑定处理后的电影数据	224
10.13 http 和 https 在小程序中的使用说明	226
10.14 跳转到更多电影页面	227
10.15 编写 movie-grid-tpl 模板	229
10.16 编写“更多电影”页面	231
10.17 实现页面下拉刷新的“三部曲”	234
10.18 在模拟器中可执行下拉刷新但在真机中无法执行下拉刷新的常见错误	237
10.19 json 中的 backgroundColor 配置的是哪里的颜色	238
10.20 实现上滑加载更多数据	239
10.21 动态设置导航栏 loading 图标	241
10.22 电影搜索	244
10.23 电影详情页面	249
10.24 电影详情页面的骨架和样式	251
10.25 编写电影详情页面的业务逻辑代码	258
10.26 预览电影海报	261
10.27 设置电影页面的导航栏标题	262
第 11 章 设置	264
11.1 设置页面	265
11.2 获取用户基本信息	272
11.3 数据缓存的异步操作	275
11.4 获取系统信息	277



11.5	获取网络状态	281
11.6	获取当前位置信息与当前速度信息	282
11.7	使用微信内置地图查看位置信息	283
11.8	监听罗盘数据制作一个简易指南针	284
11.9	在小程序中实现摇一摇	286
11.10	扫码	289
11.11	获取小程序页面二维码	292
11.12	下载并预览 pdf、word 等多种类型文档	293
第 12 章	开放接口	300
12.1	准备工作	301
12.2	用户登录	301
12.3	用户信息校验	307
12.4	解析用户加密数据获取 openId 及 UnionId	313
12.5	模板消息	316
12.6	form 表单及 picker 组件	321
12.7	发送模板消息	323
12.8	微信支付	328
12.9	真实的微信小程序登录状态维护	336
第 13 章	杂项	338
13.1	wx:key	339
13.2	scroll-view 组件：在 js 中控制滚动条	343
13.3	深入理解小程序的单向数据绑定机制	348
13.4	深入理解 scroll-view 组件的 bindscrolltolower、lower-threshold 属性	349
13.5	微信小程序发布流程	350



第 1 章

微信小程序简介

本章我们将介绍小程序的一些基本概念及特性，让大家在正式学习小程序开发前，对小程序有一个整体的认知。当然，本章你也可以完全跳过，因为它不会涉及任何“代码”方面的内容。但笔者还是建议各位读者能够耐心地看一下，技术在这个时代正在以目不暇接的速度日夜更替，尤其是 Web 技术，所以了解技术更迭的原因和意义也很重要。

1.1 什么是微信小程序

什么是微信小程序：小程序是一种不需要下载安装即可使用的应用，它实现了应用“触手可及”的梦想，用户扫一扫或者搜一下即可打开应用。也体现了“用完即走”的理念，用户不用关心是否安装太多应用的问题。应用将无处不在，随时可用，但又无须安装卸载。

张小龙

微信之父张小龙用这段略带文艺气息的描述给小程序做了定义。但小程序无论从技术上还是从理念上都不是一个新事物：从技术上讲，它借用了 React Native 的一些概念，定义了一套微信自有的组件并根据运行环境的不同（PC、iOS、Android）将这些组件编译/转化为对应平台的可运行组件；从理念上讲，百度早年的“轻应用”、QQ 右下角的“应用宝”还有支付宝里的各类小服务，早已是小程序的雏形。

我们来直观地感受一下小程序，如图 1-1~图 1-4 所示。



图 1-1 电商类小程序（图片截取自小程序京东购物）



图 1-2 服务型小程序（图片截取自小程序猫眼电影）

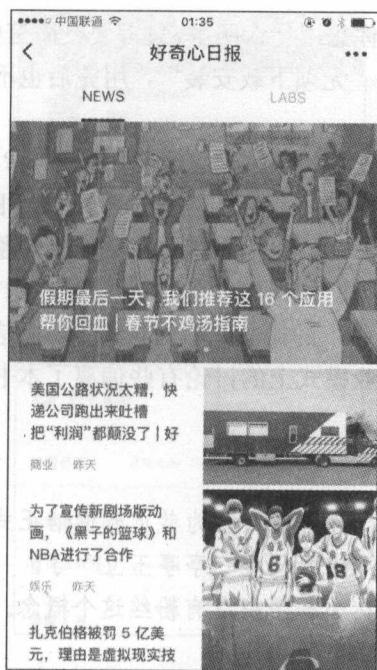


图 1-3 内容型小程序 (图片截取自小程序 Qdaily)



图 1-4 视频类小程序 (图片截取自腾讯视频)

我们本以为小程序确实就像张小龙所说的, 只适合去做用完即走的服务。但事实是, 各种类型的应用五花八门, 早已超出了公测时大家都普遍认为的小程序只适合做低频类的应用 (内容型应用是内侧时普遍认为最不适合做成小程序的一类应用), 甚至腾讯官方直接首发了一款体验非常不错的腾讯视频小程序 (这是哪门子的用完即走?)。

所以, 只有你不敢做, 没有你不能做 (游戏类小程序目前是不能做的, 直播类的有待商榷)。官方所谓的“用完即走”只是一个建议, 并不是约束。但我们要清楚地认识到, 能不能做和合不合适做是两个概念, 这取决于小程序版本的各类应用相比原生 App 到底有哪些优势。可能是成本上有优势, 可能是推广上有优势, 也可能是同微信原生功能结合有优势, 这需要开发者均衡考虑。本书还是以官方所谓的“用完即走的轻服务”为基本出发点进行论述。

有人会质疑, 这和我们常用的 App 并没有什么区别。确实如此, 小程序并没有逃脱 App 的范畴。事实上, 小程序也只是 App 的一种。那小程序的优势在什么地方?

我们试想一下, 如果上图中的这些应用并不是“寄生于”微信中, 而是以原生 App 的形式存在于 AppStore 或谷歌的应用市场中, 而我们每次使用都需要经历“打开 AppStore”→“搜索应用”→“点击下载应用”→“安装应用”→“使用应用”, 这个过程是相当烦琐的。关键是, 很多应用我们并不会经常使用, 可能一个月甚至一年才会使用 1 到 2 次, 而这些“低频”的应用却要长期地“驻扎在”我们的手机中, 需要我们去管理这些应用的“生命周期”, 这个体验是不太好的。小程序要的就是“随时可用, 触手可及”, 而现在的 App 太重了。

小程序的出现就是希望用户不用安装那么多的 App, 你想用什么服务, 只需要在微信中“扫一扫”或者“搜一下”即可享受到“触手可及”的服务, “无须下载安装”, 用完后也不需要管理它, “即用即走”。

讲到这里, 你可能对“小程序”有一些失望。并不是那么“惊天泣地”, 不是吗? 确实是这样, 如果不是腾讯已经成为社交领域独一无二的寡头, “小程序”这样的开放平台很难付诸实现(我们之前提到过百度的轻应用, 知道的人有多少?), 也不会像现在这样引爆整个互联网圈儿。但在一个超级 App 中做一个“应用市场”(虽然腾讯极力否认小程序是应用市场, 但不可避免它拥有“应用市场”的特质), 向着“操作系统”的概念“策马奔腾”, 微信确实是开创了先河(至少是首先付诸行动的)。关于小程序商业模式上的讨论有些偏离了本书的主旨, 这里就不再展开细讲了。

小程序还有以下若干“别具一格”的特性。

- 小程序没有“官方的”应用市场, 为什么强调是“官方”? 因为截止到微信正式开放小程序不到一周的时间, 不少第三方的小程序应用市场已经“亭亭玉立”了。
- 和微信公众号不同, 小程序不能被关注。所以, 在小程序中没有粉丝这个概念。
- 小程序没有群发消息和主动推送消息的能力。
- 微信中没有明显的小程序入口, 但是发现有一个【小程序】栏, 它会记录用户曾经用过的历史小程序。这些小程序的排序也没有所谓的排名, 用户最近使用的小程序排在最前面。要注意, 很多开发者找不到这个【小程序】栏, 可能的原因有以下两个: 一是必须在微信中使用过小程序, 这个【小程序】栏才会出现; 二是开发者使用的微信版本不支持小程序, 建议升级到最新版本的微信。
- 获取和使用小程序的途径只有以下几种:
 - 扫描二维码、有限能力的搜索、从发现中的【小程序】进入并打开曾经使用过的小程序以及来自于朋友或者群聊分享的小程序。注意只能扫描二维码, 不能长按二维码。
 - 之前是可以支持二维码长按进入小程序的, 这会造成大量的二维码被附加在公众号里用来传播, 所以微信干脆将“长按二维码”进入小程序的功能给屏蔽了(未来微信可能会再次放开“长按二维码”这个限制)。
- 小程序名称不能和公众号相同。
- 有一定的分享能力, 可以分享给朋友和群聊, 但不能分享到朋友圈。
- 同一个公司的公众号和小程序可以相互发现。公众号里可以看到该公司的小程序, 同时, 小程序里也可以看到该公司的公众号。点击后可相互进入到小程序或者公众号中, 如图 1-5 ~ 图 1-7 所示。

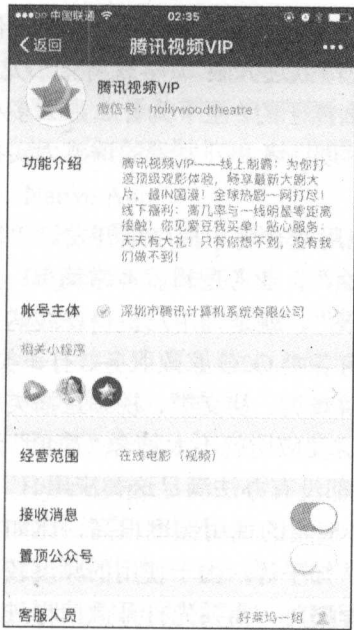


图 1-5 腾讯视频 VIP 公众号关联了 3 个小程序

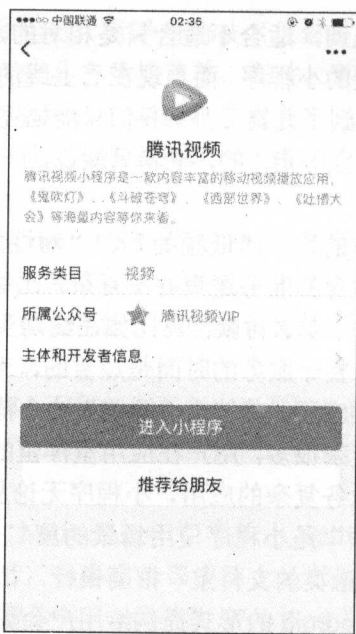


图 1-6 腾讯视频小程序关联了 1 个公众号

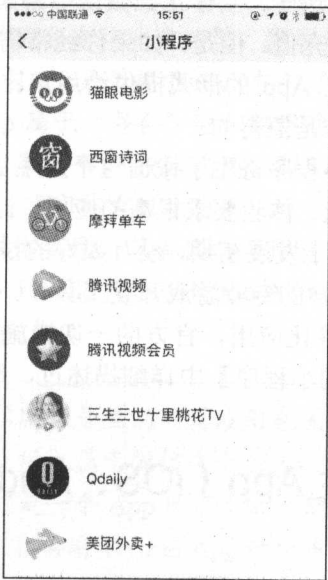


图 1-7 微信中记录用户历史小程序的列表

1.2 什么类型的应用适合用小程序开发

2016 年 1 月 11 日，张小龙在微信公开课中首次公布了“应用号”，而在 2016 年 9 月 21 日，微信正式开始了“应用号”的内测，但是却改称为“小程序”，据说是苹果不让叫“应用号”。“小程序”这个名字确实非常符合张小龙对于这类应用的定义。

再次强调, 适合不适合只是相对的, 除了游戏和直播, 微信官方并没有明确限制你不能做什么类型的小程序。而且现在已上线的小程序已是“乱花渐欲迷人眼”, 早就将张小龙的“用完即走”抛到了九霄云外。我们只能描述它本来应该是什么样子的, 至于为什么那么多小程序都不是“用完即走”的, 笔者只能说, 一个刚起步的生态对创业者和开发者的诱惑力实在是太大了。

“简单的”、“低频的”、“对性能要求不高的”应用适合用小程序来开发。“简单”是指应用本身的业务逻辑并不复杂, 比如外卖应用“饿了么”, 业务逻辑就非常简单: 挑选想吃的菜肴, 下单、付款; 再比如在线购买电影票应用“猫眼”, 就是为用户提供在线购买电影票的服务, 整个服务的时间是短暂的, “买完即走”。还有各类 O2O 家政服务、打车类应用、天气预报类应用, 都符合“简单”这个特性。相反, 一些游戏类、社交类、视频直播类应用则业务相对复杂很多, 用户在应用里停留的时间也会较长, 这类应用就不太符合“简单”这个特性。对于业务复杂的应用, 小程序无论从性能上和体验上都没有办法满足这类应用。

“低频”是小程序使用场景的第二个特点。如果某种应用的使用频度很高, 比如社交类的 QQ, 金融类的支付宝、招商银行, 社区类的百度贴吧、知乎等, 由于使用的频度较高, 以 iOS 或者 Android 的形式提供给用户会更好。当你使用小程序时, 需要先打开微信再进入小程序, 这对于高频的应用并不是太方便。小程序目前来看, 还是适合做一些如手机充值、电影购票这类使用频度不高的服务。举个例子, 当你正在微信中聊天, 突然想起手机没话费了, 那么就“顺手”点开小程序为你的手机充值。但是, 如果你经常需要特意地去寻找某类应用, 又长期需要这类应用, 那么还是以原生 App 的形式提供给用户比较好, 用户为这类应用去下载、安装和管理这个应用的一系列操作是值得的。

“对性能要求不高”是因为小程序寄生于微信这个原生 App 中, 又受限于 Web 技术的性能制约, 注定它无法去开发对性能、体验要求很高的应用, 比如“保卫萝卜”、“阴阳师”等这类游戏应用。此外, 小程序还处于发展初期, 没有太好的技术和工具去支持这类复杂度较高的游戏(没有类似于 Unity 3D 这样的专业游戏开发工具)。

小程序特别适合做线下的场景化应用, 官方的一切措施都是为了将小程序导向线下, 关于这一点, 笔者已经在【我眼中的小程序】中详细描述过, 这里就不再赘述了。

1.3 小程序与原生 App (iOS、Android) 的优劣对比

前文讲过, 小程序也属于 App 的一种, 那么它和我们现在流行的原生 App (iOS、Android) 相比, 有什么区别和优势呢?

首先, 从技术上来讲, 目前 App 的主流开发方式有三种: Web App、Native App 和 Hybrid App。我们举几个例子来看看:

- Web App

在微信“发现”里面有一个“购物”入口, 点击进去打开的是京东的移动购物页面, 这个页面实际上就是一个 Web App。支付宝的众多小服务也是 Web App, 还有“海底捞”在微信中的排号应用, 这类 App 其实就是我们经常在 PC 上浏览的网页, 只不过

加入了响应式的设计让它适合在移动端显示和运行，所采用的技术依然是 JavaScript、CSS 和 HTML。相对于其他两种 App，Web App 具有开发简单、高效，更新灵活、跨平台，大量的网页应用稍作调整即可放在移动端运行。但缺点与优点并存，Web App 性能、体验极差（对，是极差），无法使用照相机、系统通知、本地缓存等原生特性。

- Native App

也称为原生 App。这种 App 不是采用 JavaScript、CSS 及 HTML 开发，而是使用 Objective-C（iOS）或者 Java（Android）开发。微信、支付宝、斗鱼 TV 等都属于这类 App，是目前主流的开发方式。Native App 具有性能、体验非常好，组件支持完善、接口丰富等特点。但 Native App 最大的缺点在于，不能跨平台，有多少个平台就要开发多少个版本，现在主要有 iOS 和 Android 两个主流平台，还好 Windows Phone 已没了踪影。

- Hybrid App

也称为混合式 App。Hybrid App 看上去像一个 Native App，但实质上 Native 技术在这里只是作为一个容器，将 Web App 包裹了起来，在容器内部实质上运行的还是网页。Hybrid App 更像是 Web App 与 Native App 的混合体。与纯粹的 Web App 相比，Hybrid App 会有一部分访问原生组件（相机、加速器）的能力。事实上，目前主流的应用中，纯粹的原生 App 很少，绝大多数都属于混合式 App。比如，我们常见的京东、淘宝等电商类 App，由于商品及业务变化非常频繁，需要经常调整，所以这类 App 的主要页面都是采用 Web 技术来构建，只是用 Native 包装了一下。那我们如何界定，哪些 App 属于“原生”，哪些 App 属于“混合”呢？答案是：看 Web 页面在 App 中所占的比例，如果绝大多数页面都采用 Web 技术构建，那么我们称为混合式 App；而如果只有少数页面采用 Web 技术，我们称为原生应用。举个例子，今日头条这类新闻应用中绝大多数页面都采用原生技术实现，笔者倾向于称它为 Native App；而对于淘宝、京东等 App，笔者更倾向于它是 Hybrid App。Hybrid App 具有接近于 Native App 的体验、开发效率高、跨平台等特性。

有一些开发者认为微信服务号里的网页应用也属于 Hybrid App，这种说法也不无道理。因为这些网页应用也属于微信这个 Native 应用的一部分，同样运行在微信内置的浏览器中，但这是一个 App 所有者的问题。对于微信，确实是 Native App 中加入了部分“网页”，具有 Hybrid App 的特点。但我们上面讲到，目前主流 App 里很少有纯粹的 Native App，是不是算作 Hybrid App，应该看 Web 页面在 App 中的所占比例。微信是一个以社交为主要业务的 App，微信中的绝大多数核心社交与聊天功能都是原生的，所以我们还是称微信为 Native App。但是，对于服务号应用的开发者，微信并不是开发者开发的，开发者只拥有其服务号。而且服务号应用所用到的所有技术都只局限在 Web 技术里，从这一点来讲，服务号的应用应该归属于 Web App 的范畴。

此外，我们说 Hybrid App 具有一部分可以访问原生设备组件的能力。微信的 JS-SDK 确实提供了一些如拍照、录音、扫一扫等功能的接口，但相比于其他 Hybrid App 能调用的原生功能，实在是有限。从这个角度来讲，也应该将这些服务号的应用归属到 Web App 中。

其实，到底归属于什么并不重要。互联网技术中的概念层出不穷，对很多事物的定义本来就不是很明确。这里用一些篇幅解释 3 种主流类型的 App，是希望大家在对比小程序和其他类型 App 时，能有一个较完整的知识背景。

那么小程序属于以上 3 种的哪一种？严格意义上来说，它不属于以上 3 种中的任何一种，在实现技术上小程序同传统的 Hybrid 还是有很大的不同的。小程序采用 JavaScript 和 CSS 这类常见的 Web 技术开发，但它又不使用 HTML，它同 Web 没有直接的联系。小程序实际上是将一系列自己定义的组件编译成了对应平台（iOS、Android、PC）的相应可运行组件，以提高运行性能。如果一定要将小程序归并到以上 3 类 App 中，可能 Hybrid App 更合适：非原生，但使用到了 Web 技术（JavaScript 和 CSS）。

相比于 Native App，小程序具有 Hybrid App 的一些优势：

- 跨平台（对于 iOS 和 Android 两个平台只需要开发一套程序）。
- 具备接近于 Native App 的体验（注意只是接近）。
- 对原生组件有访问能力。
- 具备缓存能力。
- 上手容易，开发逻辑较为简单。

同时，小程序还具有一些它独有的特点：

- 小程序在设计时就做了很多约定式的规范：比如简单的文件结构、默认的文件命名、内置好的 Tab 栏与导航栏等，这让小程序的初学者更容易上手和理解。
- 开发环境很干净，你无须安装任何除开发工具外的其他软件。当然现在这个工具简陋的可怕，很多常见的 IDE 功能都不具备，但相比于其他 Hybrid App 的环境要求，小程序这点真的很棒。
- 发布和部署流程非常简单，几乎是“傻瓜式”，点击几下就可以将应用发布到腾讯云。
- 小程序之所以在公布后引起了互联网圈儿和开发者们极高的关注度，原因并不在技术上，无数开发者、创业者看中的是微信天然的关系链与获客能力。这也是小程序最大的优势。

但是，世间没有完美的事物，计算机世界里也没有完美的技术，你以为的优势在另一方面却成了缺点。我们一起来看下：

- 小程序为了简化复杂性，做了一些 UI 上的设计规范，确实方便了很多对 UI 要求不高的应用。但这也限制了那些对 UI 要求极高的产品发挥。
- 小程序很遗憾地不支持现有的 HTML DOM 结构，而是自己给出了一系列的组件，造就了一个封闭的开发环境，这直接导致了现有的经典 JavaScript 框架、类库都无法使

用。小程序现在的生态几乎是荒芜一片，等待着开发者们去耕耘（挑战与机遇并存，正因为没有，才有机会）。如果你想用小程序实现一组图形来展现股票或者天气的曲线，目前来看，相当烦琐。你无法使用经典的 echart 或者 highchart，你只能自己用 Canvas 来一点点地绘制。

- 截止到笔者编写本书时，小程序还不支持 WebView，这是相当头疼的一个问题。现在在很多新闻类型的应用，都是将文章数据静态化成 HTML 存储在服务器或者是 CDN 中，然后再利用 WebView 直接加载这个 HTML 来显示。不支持 WebView 直接导致了很多内容型应用没办法加载已存在的大量 HTML 页面。内容型应用现在大量的静态化页面需要被转化（已有一些第三方的组件实现了 HTML 转 WXML，基本思路是用正则表达式替换 HTML，但效果并不能让人满意）。至于微信会不会官方支持，这个很难抉择。不支持 WebView 对现在的静态化 HTML 页面是致命的打击；但兼容 WebView 就意味着在小程序里你还可以运行 Web App，而 Web App 很难去监管，性能体验也不够好，这对于小程序的发展是不利的。也许开放一个只解析 CSS 不允许运行 JavaScript 的 WebView 可能是个不错的选择，微信如何平衡这个问题，我们拭目以待。
- 小程序只实现了模板化并没有实现自定义组件，这是最令人不满意的地方。如果我们想实现一个自定义逻辑的组件，通常希望把这个组件的标签、样式以及业务逻辑打包在一起，然后可以放在项目中多个地方使用。外部客户端调用组件时，只需要传入组件所需要的参数，由组件自己来完成数据获取、转化、绑定并和 UI 层通信等操作。但小程序里的 template 只能将标签和样式（WXML 和 WXSS 文件）提取出来作为一个“模板”，却无法把组件的业务逻辑（js 文件）放在一起。也就是说，组件的业务逻辑不能够写在组件的模块儿中，只能写在“调用”组件的业务代码中，这就无法很好地复用组件的业务代码。原因我们会在后面讲到模板“template”时再来详细讲解。

我们用表 1-1 来对比小程序和现在主流App的优劣势。

表1-1 小程序和现在主流App的优劣对比

属性	微信小程序	iOS、Android
相关基础语言	JavaScript 和 CSS	Objective-C、Java
性能	较好	极好
成本	低	高
开发效率	低（目前）	较高（目前）
开发环境配置	极为简单	较复杂
新手入门速度	快	慢
适合的应用	业务简单，使用频率不高	业务逻辑复杂，使用频率高
优秀第三方组件	没有	极为丰富
新版本审核周期	较短	较长
社区支持	没有	较多

1.4 小程序会淘汰原生 App 吗

不会。连 Hybrid App 都无法撼动 Native App 的地位，更何况小程序本身只是 Hybrid App 的一个子集，运行在微信这个 Native App 之下呢？除了 Hybrid App 本身与 Native 技术的差距，微信对小程序还附加了诸多限制，比如安装包大小不能超过 1MB，不能做直播类和游戏类应用等。

小程序的定位也非常明确——做低频和业务逻辑不复杂的应用，原生 App 与小程序之间更多地将是一种互补的关系，绝对谈不上取代。

预计小程序将是 MVP 产品的践行地，是一个快速试错和调整产品思路的平台；同时绝大多数的产品也将推出自己的小程序，将全部或部分功能移植到小程序上占领更多的入口。一个平台是否流行，真的不取决于技术本身是否优秀，更多的时候开发者需要一个“理由”，无论是技术上还是商业上的理由，而微信海量的用户、全新的应用市场就是微信给开发者的一个理由。公正的评价，小程序确实在技术上无创新，但也的确具备巨大的商业价值。

1.5 Web 前端的未来

相比于 iOS 和 Android，Web 前端对技术的需求度确实要高出很多。iOS 和 Android 是为移动端量身定做的系统，移动端的局限性本身就决定了它很难作为主要的生产力工具。虽然这些年有很多移动端 App 致力于将手机变成生产力工具，但诚实地讲，如果自然语言或者其他更先进的人机交互方式不出现，移动设备很难成为“名副其实”的生产力工具。“PC 已死论”、“微软已亡论”流传了好多年，可无论是 PC 还是微软现在都活得很好。这其中一个原因还是在于人类不能只消费，还需要生产，而 PC 作为信息化社会的主要生产力工具，这是移动端无可替代的。

我们在移动端更多的时候是去用眼睛“看”，而我们在 PC 端更多的是用手去“操作”。从信息的角度讲，移动端主要负责信息的输出，而 PC 端主要负责信息的输入。有过开发经验的朋友应该体会到，相对于纯粹的显示，有“输入”操作的应用开发难度和开发成本是要高很多的。目前市场上对于 iOS 和 Android 开发者的需求量已经接近饱和了，而 Web 前端由于其本身的特性，优秀的开发者相对偏少，市场的需求量是巨大的。即使一个公司的产品以 App 为主，也不可能缺少 Web 前端开发者：

- 我们之前讲过，混合式 App 是现在的主流 App，一个 App 很难只用 Objective-C 或者 Java 来开发，必然会有 Web 技术介入。你只看几乎所有应用都有“分享到微信、QQ、微博”（分享的内容大多数都是一个网页）等功能就知道，Web 技术是不可或缺的。
- 大多数移动端应用也都有一个对应的 Web 网站。
- 现在的公司做营销和推广都离不开微信，无论是 H5 页面还是做微信服务号、企业号都是纯粹的 Web 技术。



在小程序推出之前，Web 技术在移动端时代更像是一个“黏合剂”，无处不在却又不能够独立地承担移动端的开发。Web 技术和 Web 开发者一直处于移动时代的边缘，不能没有它们，却也无法自成一体。但以 React 为代表的 React Native 和微信小程序的出现，给了 Web 开发者希望。虽然在性能体验上依然不及原生应用，但已经相当地接近了。Web 开发者终于可以在主流的移动平台中占据一席之地了。

笔者常常惊叹于 JavaScript 顽强的生命力，作为最早的浏览器交互语言，当初仅仅被当作一个玩具。这个玩具却经历了漫长的 Web 时代，抗住了 Adobe 的 Flex 和微软 Silverlight 这些所谓“富客户端应用程序”技术的猛烈攻势，不仅在移动端时代没有消亡，反而“溜到”了“后台”，以 NodeJS 的形式开始了自己的服务器之旅，现在又不“甘心”蜗居在 Web 的两端（浏览器和服务器），反而以混合式的技术形态强攻移动端。这个被 Brendan Eich 用了 10 天设计出来的脚本语言，以惊人的生命力横贯 Web 和移动时代，甚至还可以以“寄宿”的方式成为一个桌面应用程序。JavaScript 在很长一段时间，由于设计时间太短，细节考虑的不够严谨，导致 JavaScript 都是“程序混乱”的代名词，但即使这样，也无法阻挡它前进的脚步。

随着 ES2015 的普及，JavaScript 变得更加完善与强大。而现在 Web 前端的发展呈现出的是一种百花齐放的姿态，发展与更新速度远超服务器技术的更迭速度。好事还是坏事，大家各抒己见，没有定论。但有一点可以确定，Web 前端开发将是一个非常具有挑战和想象力的工作，如果你刚好走在前端开发的路上，那么恭喜你，你正行走于时代的技术浪潮上。

1.6 Web 前端开发者与小程序

招聘网站上常见的技术类职位有：iOS、Android、Java、.Net、Web 前端、DBA、大数据等。

有没有可能未来会出现小程序开发者工程师这个职位？

除了专业做微信开发的公司，小程序工程师这个职位在短期之内不会成为独立的一类职位，绝大多数的小程序将由 Web 前端工程师来开发。未来，你将看到的 Web 前端岗位要求会添加一句话：熟悉微信小程序开发者优先。正如现在的 Web 前端职位都要求应聘者精通 jQuery、熟悉 AngularJS、Grunt 等一样，小程序也将是 Web 前端职位的一项加分项。

微信小程序更多的是 App 或者 Web 网页的另一个流量入口，但绝不会替代原生 App 或者 Web 网页（至少很长一段时间内是这样，未来小程序怎么发展还有待观察）。正如我们描述的微信对现在公司营销的重要性，小程序也将成为 Web 前端开发者应该掌握的一门技术。

笔者认为，每个 Web 前端开发者都应该至少了解一下小程序，可以不精通，但至少应该写一个简单的 Demo，跑一跑，完善一下自己的技术栈和知识体系。你完全不需要把小程序当作一个平台或者是生态，你只需要把小程序当作和 jQuery、AngularJS 一样的一个 JavaScript 类库或者框架来学习即可。反正现在的前端框架多得泛滥，我们 Web 前端开发者也都在不断地开发和学习，多学一个体验接近于原生 App 的技术，何乐而不为呢？

1.7 MINA 框架与微信小程序

MINA 是官方小程序的内部开发代号，也是小程序运行框架的别名。据说 MINA 有 MINA Is Not App 的意思。到目前为止，许多开发者并没有正确理解什么是微信小程序，它和我们在网页开发中常用的 AngularJS 和 Vue 又有什么区别？

微信小程序并不是一项技术或者一个框架，微信小程序是一个生态，与之对应的应该是 iOS 生态和 Android 生态，其中微信小程序又与 iOS 生态极为相似，它们都非常封闭，而且审核非常严格（微信小程序的审核比苹果还要严格）。而 MINA 是小程序的一个框架，它提供了小程序运行所需要的接口、模型和机制。

1.8 微信小程序 beta 测试版

自 0.11.122100 版本后，微信官方开始引入测试版本的概念。微信会提前放出下一个版本的测试版，以供开发者提前对新版本进行测试并反馈 bug。比如，在 122100 版本后的一周内，微信就提供了下一个版本的 beta 版。

测试版本的概念非常重要，可以让开发者预先知道下个版本的更新内容，建议开发者应该第一时间使用测试版本测试一下自己的小程序。

本章我们对小程序做了一个较为全面的介绍，让大家在开始开发前对小程序有一个较为全面的了解。

本章主要介绍了小程序的特性与商业价值，并没有涉及开发方面的知识。小程序是一个新的生态，开发者有必要充分了解小程序的优劣。现在的技术，条条大路通罗马，选择是多样性的，只有充分了解一个技术/平台的特性，才能做出正确的选择。

除此之外，小程序的开发相对来说比较简单，至少笔者认为学习难度和学习曲线都在 AngularJS、Vue 和 React 之下。

第 2 章

小程序环境搭建与 开发工具介绍

小程序开发几乎不需要配置任何开发环境，只需要安装微信提供的一款名为微信 Web 开发者工具的 IDE 即可。本章我们将使用小程序开发工具新建一个官方提供的示例项目，并介绍开发工具的相关界面、功能与使用技巧。

2.1 微信 Web 开发者工具下载及安装

小程序的开发工具官方名称为：“微信 Web 开发者工具”，其中并不包含“小程序”3个字。看来微信的这个 IDE 并不想仅仅只是用来开发小程序。事实上也确实如此，这款开发工具不仅仅可以用来开发小程序，还可以用来调试运行在微信上的网页以及微信 JS-SDK。

开发工具的官方下载地址为：<https://mp.weixin.qq.com/debug/wxadoc/dev/devtools/download.html>。

官方提供了3个版本的开发工具安装包：Windows 64、Windows 32 和 Mac。本书中的项目开发环境为 64 位的 Windows，所以这里选择 Windows 64 版本的安装包。这里要特别提醒的是，小程序的开发工具不支持 Windows XP 系统，这一点官方文档没有明确指出。请使用 Windows 7 或者 Windows 7 以上的 Windows 操作系统或者 Mac 来安装开发工具。各位读者请根据自己计算机操作系统的实际情况下载对应版本的安装包。下载完成后，双击运行安装包出现如图 2-1 所示的界面。

按照安装向导提示，一直到安装完成。如图 2-2 所示。

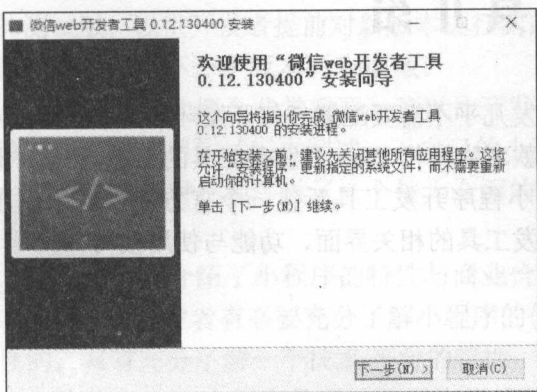


图 2-1 安装向导首页

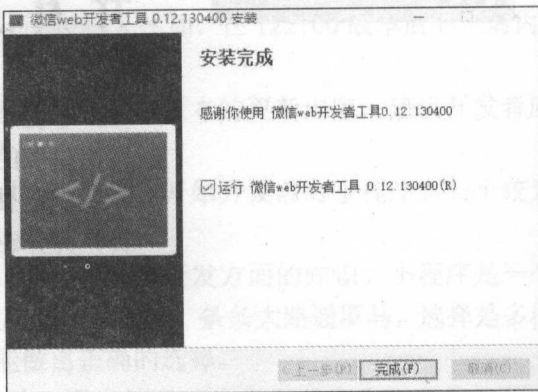


图 2-2 安装完成界面

2.2 新建第一个项目

开发工具安装完成后，我们来新建第一个小程序项目。双击打开微信 Web 开发者工具，如果你是第一次打开或者长时间未打开，开发工具都会弹出一个二维码，如图 2-3 所示。请使用微信“扫一扫”功能扫描二维码。是的，开发工具也需要登录，而登录的身份就是你的微信号。

登录后可看到图 2-4 所示的开发者工具首选页面。



图 2-3 扫描二维码登录开发者工具

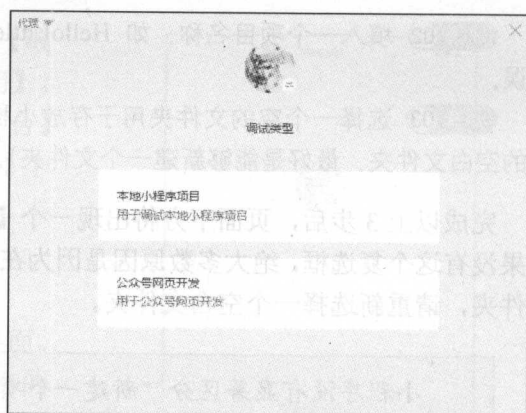


图 2-4 开发者工具首选页面

首选页面一共有 2 个工具，分别是【本地小程序项目】和【公众号网页开发】。【本地小程序项目】用来开发、调试、发布微信小程序；而【公众号网页开发】用来开发和调试微信公众号、订阅号的网页应用。由于小程序和公众号是两个完全不同的平台和体系，公众号的开发不在本书的讨论范围之内，我们只关注【本地小程序项目】这个工具，有兴趣的读者可以自行去研究学习“公众号网页开发”工具。点击【本地小程序项目】，将出现如图 2-5 所示的页面。

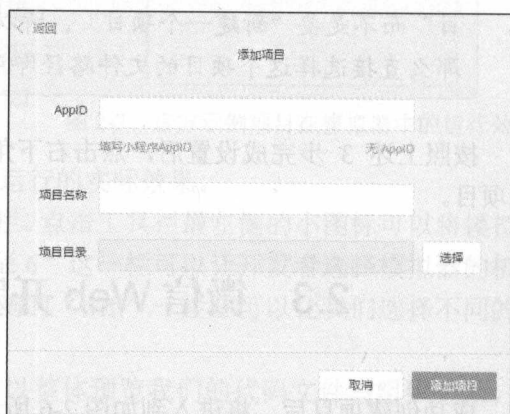


图 2-5 添加项目页面

页面中需要填入的选项有【AppID】、【项目名称】、【项目目录】这 3 个选项，我们重点看【AppID】这一项。

AppID 代表微信小程序的 ID 号，必须拥有微信小程序账号才可以申请这个 ID 号。读者可以到微信公众平台官网注册申请微信小程序账号，注册地址为：<https://mp.weixin.qq.com/>。

遗憾的是截止本书完稿时，小程序账号并不对个人开发者开放，只对企业、政府和组织开放。没有 AppID 将无法在真机上运行小程序，但并不影响我们学习小程序开发，小程序开发工具提供了一个很好的模拟器，可以在 PC 或者 Mac 下模拟小程序的运行。本书中的绝大多数功能都可以在 PC 或者 Mac 下运行，少部分无法在模拟器下运行的功能，本书也会使用图片的方式向读者展示运行效果，不用担心没有 AppID 无法真机运行小程序的问题。

既然没有 AppID，那么我们就选择【无 AppID】这个选项。【无 AppID】主要用于方便开发者学习和调试。相比于有 AppID，无 AppID 模式的运行环境非常宽松，例如，无 AppID 环境下无须配置可信任访问域名、不会校验 TLS 版本等（这些特点将在本书的后面具体来讲解，读者现在无须搞懂这些内容）。现在，我们只需要选择【无 AppID】即可。

按照以下步骤创建第一个小程序项目。

步骤 01 在【AppID】选项选择【无 AppID】。

步骤 02 填入一个项目名称，如 `HelloLittleApple`。建议使用英文，中文可能会引起一些未知错误。

步骤 03 选择一个空的文件夹用于存放小程序项目文件（注意，一定要选择一个没有任何文件的空白文件夹，最好是能够新建一个文件夹）。

完成以上 3 步后，页面下方将出现一个【在当前目录中创建 quick start 项目】的复选框。如果没有这个复选框，绝大多数原因是因为在上面的第三步里，你选择的文件目录不是一个空文件夹，请重新选择一个空白文件夹。

小程序没有显著区分“新建一个项目”和“打开一个已存在的项目”。如果你选择的目录是一个空的文件夹，则小程序会认为你是想“新建项目”；而如果你选择的文件目录不是空文件夹，则小程序会认为你是要“打开一个已存在的项目”而不是要“新建一个项目”。所以，如果你想打开一个已存在的小程序项目，那么直接选择这个项目的文件路径即可。

按照上述 3 步完成设置后，点击右下角的【添加项目】，将成功创建一个官方提供的示例项目。

2.3 微信 Web 开发者工具界面功能介绍

成功创建项目后，将进入到如图 2-6 所示的开发者工具主界面中。

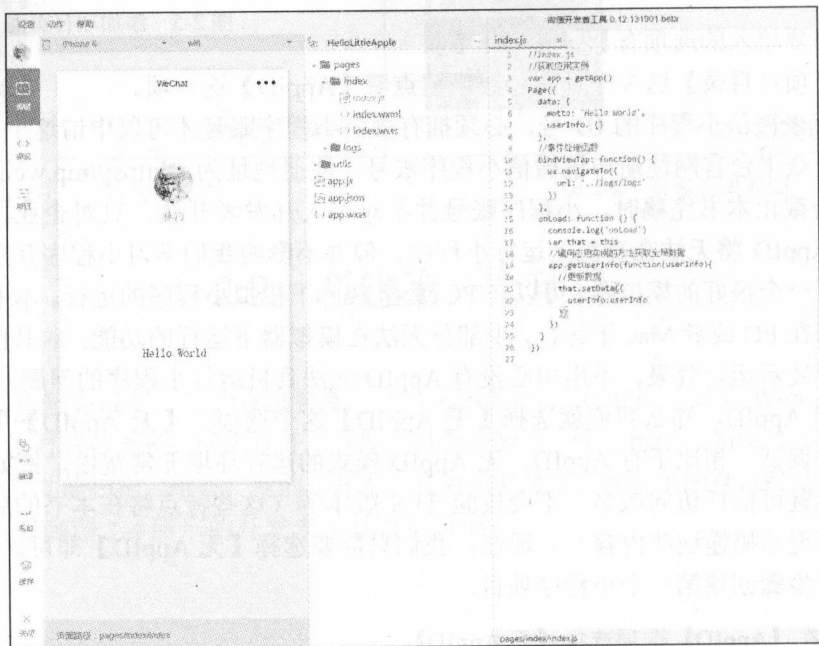


图 2-6 微信 Web 开发者工具主界面

在主界面最左侧垂直分布着 5 个选项卡：【编辑】、【调试】、【项目】、【编译】和【关闭】。当点击【调试】后，左侧下半部分还会出现【后台】和【缓存】这 2 个选项卡。下面依次来介绍这 7 个选项卡。

2.3.1 编辑选项卡

如图 2-6 所示的界面就是编辑选项卡的主界面。编辑界面分为左、中、右 3 部分。左边是模拟器的预览视图，中间是代码的树状目录，右边是代码编辑区（也就是写代码的地方）。

我们来看看左边模拟器的相关功能。左边的模拟器可以模拟微信小程序在客户端真实的逻辑表现，我们可以在这里预览到小程序的运行情况，如图 2-7 所示。



图 2-7 官方示例项目在模拟器中的运行效果

“Hello World”的这部分就是官方示例项目运行的实际效果。

在模拟器的顶部有如图 2-8 所示的一条工具栏。点击工具栏最左侧的小图标可以将模拟器关闭，再次点击则会打开模拟器；中间的“iPhone 6”这一栏可以让开发者选择模拟器的机型用来模拟小程序在不同机型上的运行情况；最右侧的“wifi”一栏，可以让我们选择不同的网络环境。

中间部分是项目文件的树状管理器，这里可以整体预览我们的代码文件与组织关系。在这部分的顶部，如图 2-9 中，有 3 个较为重要的功能：最左侧的小图标点击后会隐藏树状文件管理器；中间部分显示了项目的项目名称；最右侧的“...”小图标用来执行一系列的文件相关操作。

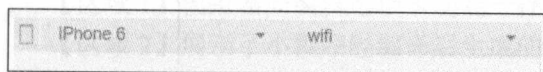


图 2-8 模拟器顶部工具栏

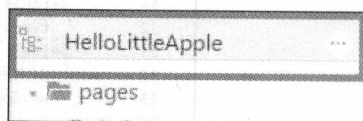


图 2-9 树状文件管理器

点击图 2-9 中最右侧的“...”小图标将出现如图 2-10 中框选的区域。

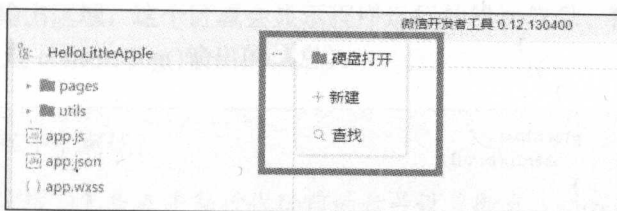


图 2-10 文件操作功能选项卡

点击【硬盘打开】将可以在 Windows 中打开项目的文件目录，点击【查找】可以搜索包含相关关键字的内容，点击【新建】将打开如图 2-11 所示的菜单。

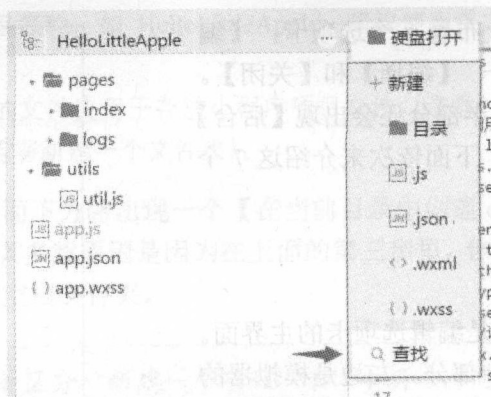


图 2-11 新建文件菜单

在新建菜单中可以新建文件夹与 4 种类型的小程序文件，关于这 4 种类型的小程序文件（.js、.json、.wxml、.wxss）将在文后具体介绍。除了可以在顶部打开新建菜单外，如果想在某个指定的文件夹内新建文件或者更改某个指定文件的文件名，可以将鼠标移动到相关文件夹或者文件上，这时同样会在鼠标悬停的位置出现“...”小图标，点击后即可打开类似如图 2-11 所示的新建菜单。

整个【编辑选项卡】的右半部分是代码编写区域，如图 2-12 所示。

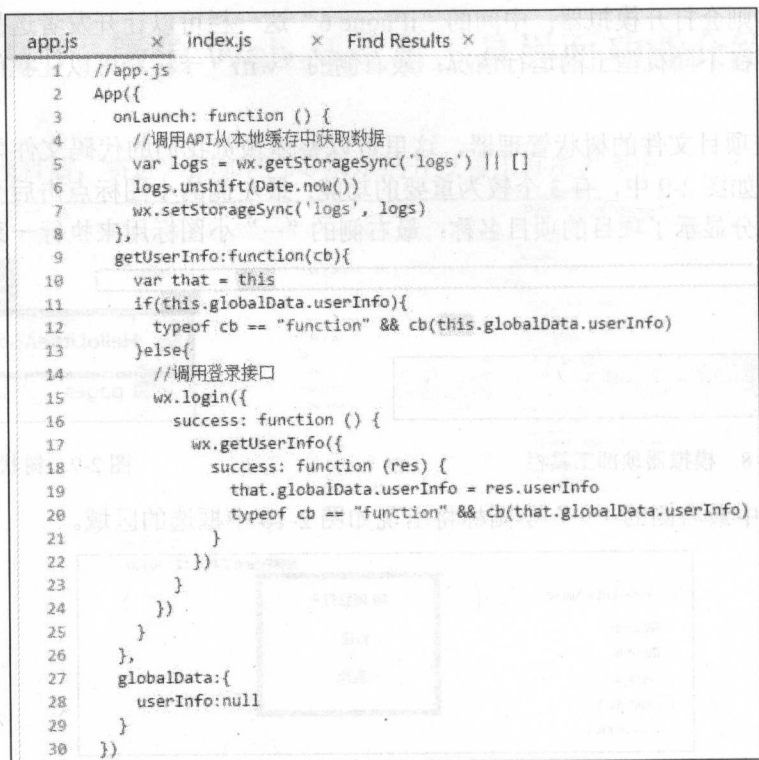


图 2-12 代码编写区域

代码编辑区域比较简单，这里就不再赘述了。

小程序开发工具目前功能比较简单,比如无法调整字体大小。这里介绍一个方法可以放大整个开发工具,从而间接实现放大字体:按住 Ctrl,并滑动鼠标滚轮从而实现字体放大或者缩小。但是这种方法是通过放大整个开发工具来实现的字体放大,副作用就是左侧的【模拟器】会出现黑色的边框。希望微信尽快完善小程序开发工具的基本功能。

2.3.2 调试选项卡

图 2-13 展示了调试选项卡的界面。

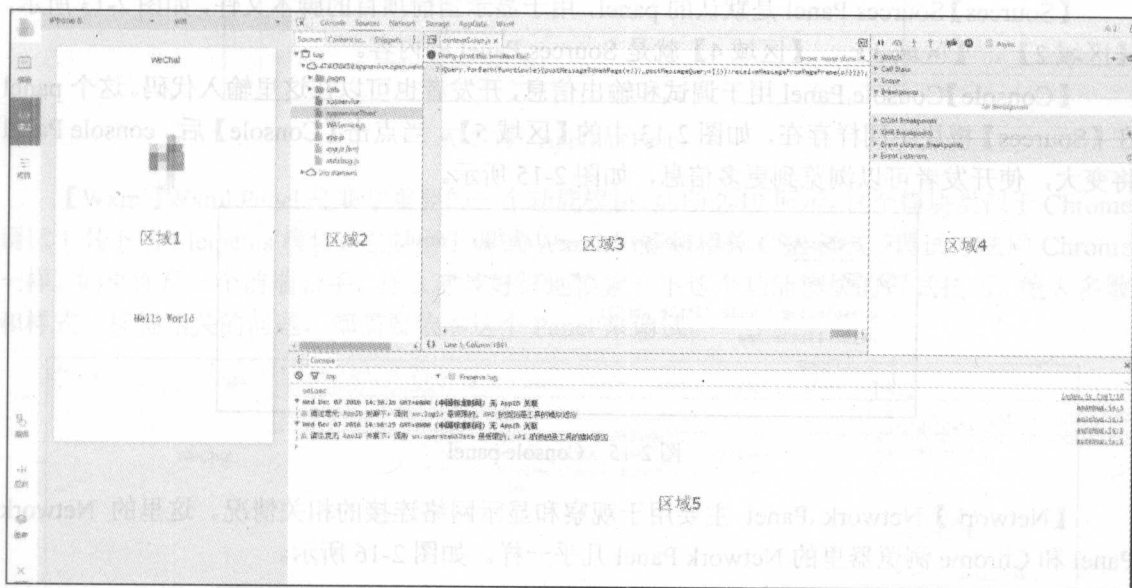


图 2-13 调试选项卡

【区域 1】模拟器同编辑选项卡功能一样,这里就不再赘述了。

【区域 2】展示了小程序经编译后生成的文件和文件结构。

【区域 3】是编译后的文件内容查看区,可以在这个区域的文件里设置调试断点。

【区域 4】调试功能区,可以在这里查看变量状态与数值、断点设置情况、变量作用域等。这个区域和 Chrome 浏览器里的调试工具几乎一样,调试的方法和快捷键也是相同的。

【区域 5】信息输出区域,这个区域会显示程序运行的错误信息、警告信息以及用户自己打印的相关信息(通过 `console.log()` 输出的信息)。

如何在小程序中调试?

可以在【区域 3】中点击每行代码前的行号设置断点。当代码运行到断点处后,将停止。常用快捷键有【F10】单步执行,【F11】进入方法,【F8】继续运行到下一个断点。更多快捷键可自行在【区域 4】中查看。

在整个调试选项卡中，最重要的部分还是【区域2】和【区域3】顶部的6个功能模块儿，如图2-14所示。



图 2-14 调试选项卡的 6 个功能 Panel

我们逐一介绍这 6 个模块儿。

【Sources】Sources Panel 是默认的 panel，用于显示当前项目的脚本文件。如图 2-13 所示，【区域2】、【区域3】、【区域4】就是 Sources Panel 的内容。

【Console】Console Panel 用于调试和输出信息，开发者也可以在这里输入代码。这个 panel 在【Sources】模块中同样存在，如图 2-13 中的【区域5】。当点击【Console】后，console Panel 将变大，使开发者可以浏览到更多信息，如图 2-15 所示。

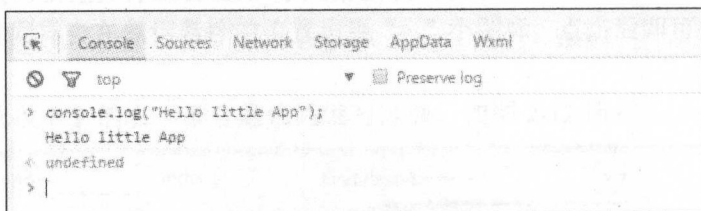


图 2-15 Console panel

【Network】Network Panel 主要用于观察和显示网络连接的相关情况。这里的 Network Panel 和 Chrome 浏览器里的 Network Panel 几乎一样。如图 2-16 所示。

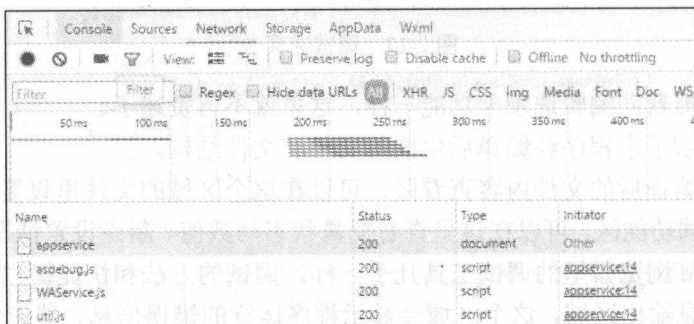


图 2-16 Network Panel

【Storage】Storage Panel 用于显示当前项目的数据缓存情况，如图 2-17 所示。关于数据缓存将在项目开发中具体讲解。

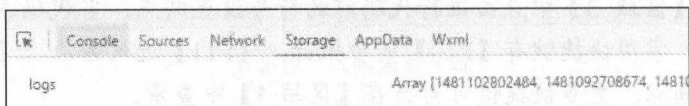


图 2-17 Storage Panel

【AppData】AppData Panel 用于显示项目中被激活的所有页面的数据情况，这些数据主要是用来做数据绑定，如图 2-18 所示，关于数据绑定我们同样放在项目开发中具体讲解。在这里不仅可以查看数据情况，还可以更改数据，小程序框架会实时地将数据的变更情况反馈到 UI 界面上。

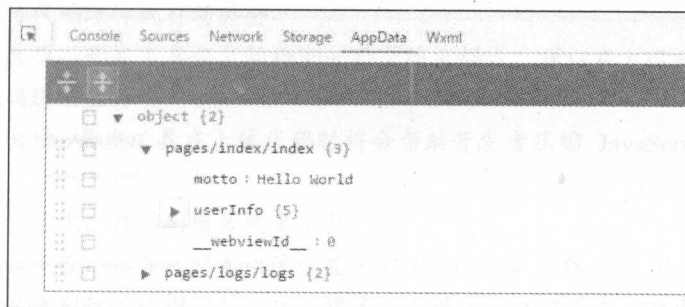


图 2-18 AppData Panel

【Wxml】Wxml Panel 是非常重要的一个功能模块，如图 2-19 所示。这个模块类似于 Chrome 调试工具下的 Elements 模块，主要用于调试 Wxml 标签和相关 CSS 样式，调试方法同 Chrome 一样。如果你是一个前端新手，那么建议好好地摸索一下这个功能模块的调试技巧，绝大多数和样式、标签相关的问题，都需要依靠这个 Panel 来调试。

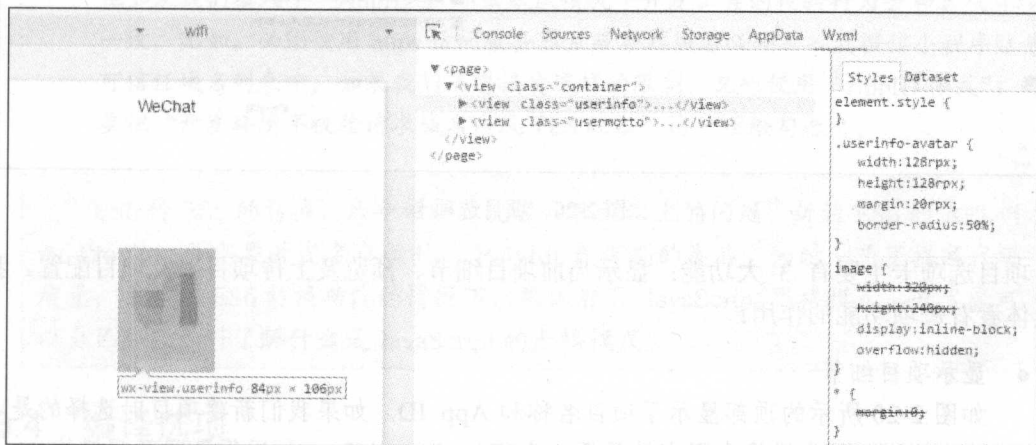


图 2-19 在 Wxml Panel 下可以同时查看 UI、标签和 CSS 样式 3 者的相关数据及关联关系

以上的 6 个模块功能对我们开发非常有帮助。如果你在开发中遇到一些稀奇古怪的问题，那么最好的解决办法就是使用这 6 个 panel 来解决。我们在后面的章节中也会经常回过头来讲解这些 panel 的使用技巧，并使用这些 panel 解决我们的问题。

以上 6 个 panel 包含的功能非常多，不需要现在就把每个 panel 的细节了解的非常清楚。笔者的建议是知道以上 6 个 panel 是做什么的以及它们简单的使用方式即可，在后续的内容中将会用实际的数据来演示如何使用这些 panel，开发者无须担心，轻轻松松地继续阅读本书吧！

2.3.3 项目选项卡

项目选项卡主要用来管理和设置项目，如图 2-20 所示。

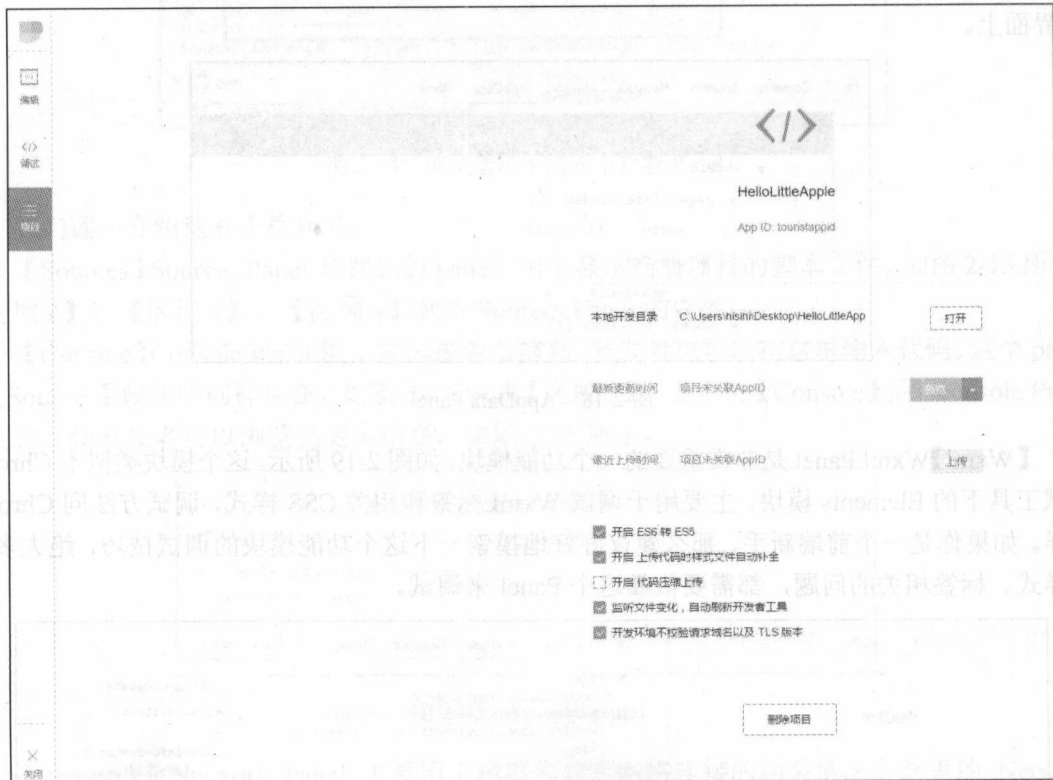


图 2-20 项目选项卡

项目选项卡主要有 3 大功能：显示当前项目细节、预览及上传项目以及项目配置。我们来具体看看每项功能的作用：

- 显示项目细节

如图 2-20 所示的顶部显示了项目名称和 App ID。如果我们新建项目时选择的是“无 AppID”，则这里将会固定地显示一个“touristappid”；如果你设置了项目的 AppID，则这里会显示项目的 AppID。

- 预览及上传项目

预览按钮可以实现在真机上运行小程序，但前提条件是你必须有一个微信小程序账号。如果你有小程序账号，则可以得到一个 AppID，新建项目时，填入这个 AppID 即可。关于小程序账号的相关配置，我们将在本书的后面章节中详细介绍。由于我们在新建项目时选择的是“无 AppID”，这里的预览按钮将处于灰色不可点击的状态。上传按钮用于上传和发布项目，上传的项目将可以在小程序账号后台页面中查看到。同样，由于选择的是“无 AppID”，这里目前也是处于灰色不可点击状态。

- 项目配置

在目前版本中，总计有 5 个配置选项，下面我们逐一介绍：



➤ 开启 ES6 转 ES5

小程序支持使用 ES6 来编写代码。如果使用 ES6 来编写代码，框架会默认使用 babel 将开发者的代码转换为 ES5 代码（这样做的主要原因是为了保持对三端：iOS、Android 和开发工具模拟器的良好兼容性）。开发者可以在项目中关闭这个选项。

➤ 开启上传代码时样式自动补全

开启此选项，开发工具会自动检测并补全缺失样式，保证在 iOS 8 上的正常显示。

➤ 开启代码压缩上传

开启此选项，开发工具在上传代码时将会帮助开发者压缩 JavaScript 代码，减小代码包体积。

➤ 监听文件变化，自动刷新开发者工具

开启此选项后，如果代码发生了改变（需要 Ctrl+S 先保存），小程序开发工具会自动帮助开发者刷新调试模拟器，从而提高开发效率。也就是说，开发者不再需要手动地点击编译按键即可实时预览小程序运行效果，这是非常好用的一个功能。

➤ 开发环境不校验请求域名以及 TLS 版本

开启此选项，开发工具将不会校验安全域名以及 TLS 版本，帮助在开发过程中更好地完成调试工作。我们前面提到过，在开发工具里如果选取了“无 AppID”模式，那么开发工具的安全限制级别非常低，不需要使用 https 访问服务器，也不会校验 TLS 版本。但如果我们填入了“AppID”，那么默认情况下开发工具的校验行为会和真机环境保持一致，比如，必须使用 https 访问服务器且服务器域名必须加入到微信小程序账号中的可信任域名列表中。如果我们不想接受这样的限制，又想使用“AppID 模式”，那么需要把“开发环境不校验请求域名以及 TLS 版本”这一项给勾选上。

ES6 转 ES5 的转换，只会帮助开发者处理语法上的问题，新的 ES6 的 API 例如 Promise 等需要开发者自行引入 Polyfill 或者别的类库。同时，为了提高代码质量，在开启 ES6 转换功能的情况下，默认启用 JavaScript 严格模式。开发者可以自己查阅资料了解什么是 JavaScript 的严格模式。

2.3.4 编译选项

严格也说，“编译”选项不是一个选项卡，只是一个功能按键。编译选项分为两种：默认编译和自定义编译。特别注意编译按键有 2 个小图标，上面的小图标是默认编译，下面的小图标是自定义编译。如图 2-21 所示。

编译快捷键 Ctrl+B。最新版本中的小程序已经增加了实时预览的功能，在更改代码后只需要使用 Ctrl+S 保存，开发工具就会自动编译、更新代码。



图 2-21 编译按键

但笔者在实际的开发过程中也发现，有时候 Ctrl+S 保存后代码运行会出现异常。这个时候，请手动点击执行一下编译。

关于自定义编译的设置，将放在项目文章详情页面一章中讲解。

2.3.5 后台选项

同【编译】选项，【后台】也是一个功能按键，可以在模拟器中模拟应用程序的前后台切换操作。以 iPhone 为例，当我们运行一个应用程序时，点击 iPhone 的“Home”键，应用程序将被切换到后台，但并没有关闭。具体的功能作用我们会在后面章节介绍小程序生命周期时讲解。

2.3.6 缓存选项

【缓存】是非常重要的一个功能选项。点击【缓存】会出现一个子菜单，包括 4 个选项：【清除数据存储】、【清除文件存储】、【清除工具授权数据】和【清除手机授权数据】。缓存选项是开发工具提供给我们用来进行数据缓存的小工具。关于数据缓存和文件缓存我们将在后面的项目章节中深入讲解。

2.3.7 关闭选项

【关闭】选项点击后将关闭开发工具，但开发工具并不会完全退出，而是会停留在项目选择页面上。

小程序现在还处于公测阶段，开发工具的 bug 还比较多。很多 bug 的暂时解决方法和我们经典的“重启电脑”类似，重启一下开发工具就好了。所以如果你遇到一些莫名其妙的现象，比如在早期的版本中，有时候某些事件会执行两次，这个 bug 当你重启开发工具后就不会再出现，你可以尝试完全关闭开发工具再开启。这里提醒一下大家，重启开发工具需要完全关闭开发工具，而点击【关闭】后并没有完全关闭开发工具，需要继续叉掉项目选择页面，才算完全退出。

2.3.8 快速打开官方 API 文档

作为一个开发者，官方 API 文档的使用会非常频繁。不同于其他开发工具【F1】是帮助选项，小程序的【F1】给出的是【查看命令面板】这项功能。小程序提供了一个快速打开官方文档的方法，即点击开发工具左上角的【帮助】→【关于】，在弹出的“关于”对话框中点击【点击打开文档】，即可马上进入官方 API 文档。

官方 API 文档除了用于经常查阅 API 外，推荐大家在版本更新后第一时间去查看更新内容。最新的更新内容对我们开发者来说尤其重要，更新内容通常会出现在官方 API 文档的【工具】→【下载】以及【历史更新日志】里。

2.3.9 开发工具的更新

微信小程序开发者工具将自动更新，无须开发者手动更新。每次启动开发工具后，如果有新版本则会自动提示下载。

在 Windows 下，如果开发者重启开发工具后依然没有自动更新，请退出开发工具，右键



点击开发工具图标，选择【以管理员身份运行】。以上解决方案在 Windows 10 下测试可行。如果以上方案都不可用，请前往官网下载并安装最新版本的开发工具。

官方在 140600 版本中指出，下一个版本才能完全修复无法自动更新的问题。

2.3.10 常用小程序快捷键

1. 格式调整

Ctrl+S: 保存文件

Ctrl+[, Ctrl+]: 代码行缩进

Ctrl+Shift+[, Ctrl+Shift+]: 折叠打开代码块

Ctrl+C Ctrl+V: 复制粘贴，如果没有选中任何文字则复制粘贴一行

Shift+Alt+F: 代码格式化

Alt+Up, Alt+Down: 上下移动一行

Shift+Alt+Up, Shift+Alt+Down: 向上向下复制一行

Ctrl+Shift+Enter: 在当前行上方插入一行

Ctrl+Shift+F: 全局搜索

2. 光标相关

Ctrl+End: 移动到文件结尾

Ctrl+Home: 移动到文件开头

Ctrl+i: 选中当前行

Shift+End: 选择从光标到行尾

Shift+Home: 选择从行首到光标处

Ctrl+Shift+L: 选中所有匹配

Ctrl+D: 选中匹配

Ctrl+U: 光标回退

3. 界面相关

Ctrl + \: 隐藏侧边栏

Ctrl + m: 打开或者隐藏模拟器

本章我们主要预览了整个微信 Web 开发者工具的全貌。笔者一向认为，工具的使用需要用具体的案例来推动，我们会在后续的项目编写中经常使用这几个选项卡和工具来解决开发中遇到的各类问题。下一章，我们将正式开始小程序的编码之旅。

第 3 章

从第一个简单的“Welcome” 页面开始小程序之旅

本章我们将正式开始 Orange Can 项目的编码工作。项目从编写一个最简单的“welcome”欢迎页面开始，并在编写页面的过程中逐步介绍小程序的基本文件结构、CSS 的使用限制、自适应单位 rpx、全局样式、App.json 配置文件以及 Flex 布局等小程序开发的必备知识。

3.1 认识小程序的基本文件结构

我们还是以第2章中新建的官方示例项目 HelloLittleApple 为参考，来看一下构成一个小程序的基本文件要素，如图3-1所示。

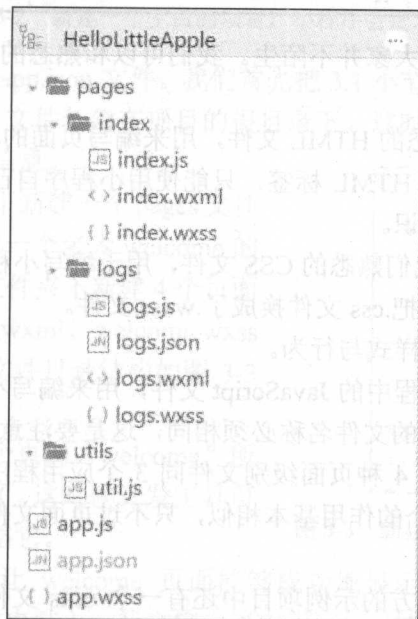


图 3-1 官方示例项目的文件及文件结构

不同于其他框架，小程序的目录结构非常简单，也非常易于理解。

首先我们看到根目录下面有3个文件：app.js、app.json 和 app.wxss。一个小程序项目必须有这3个描述 App 的文件，它们必须放在应用程序的根目录下，否则小程序会提示找不到 app.json 文件。表3-1描述了3个文件的意义。

表 3-1 app.js、app.json 和 app.wxss 的含义

文件	必填	作用
app.js	是	小程序逻辑文件
app.json	是	小程序配置文件
app.wxss	否	全局公共样式文件

这3个文件是应用程序级别的文件。

接着是和这3个应用程序级别文件平行的 pages 文件夹。一个小程序由若干个页面文件构成，比如图3-1中 pages 文件夹下就有2个页面，分别是 index 页面和 logs 页面。每个页面可以由4个文件构成，分别是：.js、.wxml、.wxss 和.json 文件。表3-2描述了这4个页面文件的意义。

表 3-2 .js、.wxml、.wxss 和.json 文件的含义

文件类型	必填	作用
js	是	页面逻辑
wxml	是	页面结构
wxss	否	页面样式表
json	否	页面配置

其实，这 4 个文件的作用大家并不陌生。我们可以和熟悉的 Web 前端开发技术做一个对比。

wxml 文件类似于我们熟悉的 HTML 文件，用来编写页面的标签和骨架，不同的是 wxml 文件里的标签元素不可以使用 HTML 标签，只能使用小程序自己封装的一些组件，这些组件也是我们后面要重点学习的知识。

wxss 文件的作用类似于我们熟悉的 CSS 文件，用于编写小程序的样式，实际上小程序的样式编写语言就是 CSS，只是把.css 文件换成了.wxss 文件。

json 文件用来配置页面的样式与行为。

js 文件类似于我们前端编程中的 JavaScript 文件，用来编写小程序的页面逻辑。

以上 4 种类型的页面文件的文件名称必须相同，这是要注意的一个地方。

我们可以看到，小程序的 4 种页面级别文件同 3 个应用程序级别文件相比，多出了一个 wxml 页面标签文件，其他 3 个的作用基本相似，只不过页面文件作用于页面本身而应用程序文件作用于应用程序整体。

除了 pages 文件夹外，官方的示例项目中还有一个 utils 文件夹，这个文件夹用来存放一些公共的 js 文件，比如 utils 下面的 util.js。我们可以任意定义类似于 utils 文件夹的目录，并放在小程序的任意位置，小程序对此并没有任何限制。

3.2 开始动手编写第一个小程序页面

掌握以上的少量知识，我们就可以开始编写小程序了，是不是很惊奇。是的，小程序就是这样一门适合实践的技术，让我们马上开始吧。

我们从零开始新建一个项目。每个项目都有一个自己的名字，比如 Google 的 Tensor Flow（一个机器学习项目）、淘宝的 Ocean Base（一个分布式数据库）、微软的 Azure（云计算“蔚 蓝”），还有大家写代码使用的各类框架：Flask、Spring、jQuery。我们的项目虽小，但还是要给它起个名字，就叫“Orange Can”吧。没有什么特殊的意思，纯粹是因为笔者现在想吃“桔 子罐头”了。大家可以随意来给项目命名。

我们按照在 2.2 小节里所讲的方式，新建一个项目。仍选择【无 AppID】，并且不勾选【在 当前目录中创建 quick start 项目】这个选项，因为我们要从零开始编写一个项目，所以每个文 件都将由自己亲手创建。项目创建后，会出现一个如图 3-2 所示的错误提示，这是因为现在的 项目里还没有任何文件，由于缺少必要的文件，所以小程序会报错。之所以完全新建一个全新

的项目，是为了向开发者展示这些常见的错误消息，如果大家不想经历这些错误，那么可以在官方提供的 quick start 项目上修改。



图 3-2 新建一个空白的项目，小程序会提示错误

错误信息提示我们缺少 app.json 文件。我们首先把 3.1 小节中所提到的 3 个应用程序文件 app.json、app.js 和 app.wxss 文件新建在项目的根目录下。这时候，小程序依然会提示错误信息，可以先忽略掉这些错误信息。

我们继续在项目根目录下新建一个 pages 文件夹，并在 pages 文件夹下新建一个名为 welcome 的文件夹，接着再在 welcome 文件夹下新建 4 个页面文件：welcome.js、welcome.wxml、welcome.wxss 和 welcome.json。新建后的文件目录结构如图 3-3 所示。

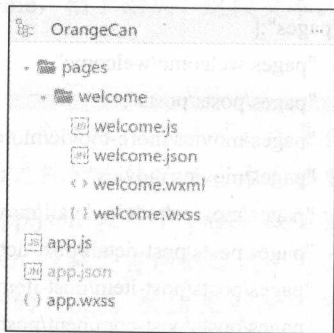


图 3-3 新建 Welcome 页面后的项目文件结构

完成以上操作后，第一个页面“welcome”所需要的全部文件就新建完毕了。这时候开发工具可能依然提示有错误，继续忽略它。

我们现在要做的事情是让 welcome 页面能够成功地显示出来。打开/pages/welcome/现 welcome.wxml 文件，在文件内敲入一行文字：“Welcome，桔子罐头”。如何让这段文本成功地显示在小程序中呢？

要显示 welcome 这个页面，必须让小程序的 MINA 框架知道这个页面的“存在”以及这个页面的具体位置（文件路径）。所以，我们需要在某个应用程序级别配置文件中注册这个 welcome 页面。那么哪个文件是用来做应用程序级别的配置的呢？回顾上一小节中所讲的内容就应该知道，app.json 文件就是小程序提供给我们的全局配置文件。

那么，我们来学习一下如何在 app.json 中注册 welcome 页面。在 app.json 中加入如下代码：

代码清单 3-1 app.json

```
{
  "pages": [
    "pages/welcome/welcome"
  ]
}
```

上面这段代码将 welcome 页面注册到了小程序中。代码是一个典型的 json 对象。这个对象的第一个属性 pages 接受一个数组，数组的每一项是一个字符串，用来指定我们的小程序将由哪些页面组成。每一项由对应页面的【路径+文件名】组成。比如上面这段代码中的 pages/welcome/welcome，就指定了 welcome 页面的页面路径。

注意，页面路径前面不要加“/”。形如“/pages/welcome/welcome”这样的路径是错误的。如果加入了“/”，小程序会提示错误：无法找到 welcome 页面。

这里要特别强调，路径最后一段 welcome，不需要指定具体的文件扩展名，无须写成 pages/welcome/welcome.wxml。MINA 框架将会自动去寻找页面路径，并将页面的.json、.js、.wxml 和.wxss 这 4 个文件进行整合。

如果有多个页面，需要将每个页面的路径加入到 pages 这个数组下，否则小程序不会加载这些页面。下面代码是 Orange Can 项目后期的 pages 注册情况。

代码清单 3-2 app.json

```
{
  "pages": [
    "pages/welcome/welcome",
    "pages/posts/posts",
    "pages/movies/more-movie/more-movie",
    "pages/movies/movies",
    "pages/movies/movie-detail/movie-detail",
    "pages/posts/post-detail/post-detail",
    "pages/posts/post-item/post-item",
    "pages/posts/post-comment/post-comment"
  ]
}
```

页面的添加或者删除都需要在 pages 数组下面增减对应的页面路径，否则小程序会报错。当然现在只有一个 welcome 页面，那么 pages 下面先加入一个页面就可以了。随着 Orange Can 项目的不断开发，我们将在 pages 下面加入越来越多的页面路径。

现在，我们的小程序可能还会出现如图 3-4 所示的错误。

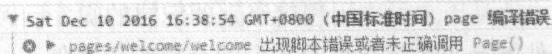


图 3-4 未正确调用 Page() 的错误提示

出现这个错误的主要原因是，welcome.js 文件是一个空文件，这是小程序所不允许的。即使我们的 welcome 页面中没有任何 JavaScript 代码，依然需要在 welcome.js 中主动调用一下 Page() 方法。我们在 welcome.js 文件中加入以下代码：

代码清单 3-3 welcome.js

```
Page({
})
```

关于 Page 方法的用法，我们将在编写 welcome.js 页面的 JavaScript 代码时具体讲解，现在只需要知道页面的 js 文件是不可以完全为空白的，否则小程序会报错。



这个时候，我们的“Welcome，桔子罐头”这段文本还是没有在小程序中正确显示。那么试着在 welcome.json 中加入如下代码：

代码清单 3-4 welcome.json

```
{  
}
```

是的，正如 welcome.js 文件不能为空一样，welcome.json 文件同样不可以为空，即使你目前不想在.json 文件中配置任何属性，也需要加入一个空的{}，以保证小程序能正确执行。

当我们完成以上所有的操作后，Ctrl + S 保存一下项目。此时，我们的小程序应该不会再报错了，同时在模拟器中也应该能够正确地显示出“Welcome，桔子罐头”这段文本。这说明 welcome 页面已经被 MINA 框架正确地加载和运行了。

我们每次创建一个新页面时，都需要手动地新建一个目录+4 个文件，这是相当麻烦的事儿。这里告诉大家一个一次创建 4 个页面文件的小技巧（官方文档里没有提到过，开发工具也没有显示的标识）。如果 app.json 文件下 pages 数组里的页面路径，指向的是一个不存在的文件，那么 MINA 框架会自动创建这个页面的 4 个文件。我们可以试一下，在 app.json 文件的 pages 数组里添加一项“pages/orange/orange”，然后保存项目，会发生什么呢？

通过这样的方式新建的页面文件将自动补全每个页面文件里必须的基本代码，不会出现错误。本小节讲解手动创建文件的例子，是为了向开发者展示一些常见的错误提示并解释错误的原因。后续页面文件的创建将采取这种比较方便的方式。

3.3 构建 welcome 页面的元素和样式

在上一小节中，我们仅仅正确地加载和显示了 welcome 页面，还没有编写任何页面代码。在本小节中，将尝试为 welcome 页面添加一些标签元素。

可以在本书的彩页中查看 welcome 页面的最终设计图。这个设计图的设计元素非常简单，仅由一张圆形的图片、一段文本和一个按钮构成。下面我们一起来完成第一个小程序页面吧。

首先来编写 3.2 小节中所创建的 welcome.wxml 文件，在该文件中键入以下代码，这段代码将 welcome 页面所需要的标签元素全部填入到页面源文件中。

代码清单 3-5 welcome.wxml

```
<view>  
  <image></image>  
  <text>Hello，桔子罐头</text>
```

```

<view>
  <text>开启小程序之旅</text>
</view>
</view>

```

这段代码总共使用了 3 个微信小程序的组件，分别是 view、text 和 image 组件。view 组件通常作为容器来使用，类似于 HTML 中的 div 标签；text 组件用来显示一段文本，类似于 HTML 中的 span 标签，本例中第一个 text 组件用来显示一段文本“Hello, 桔子罐头”；而 image 组件用来显示一张图片，类似于 HTML 中的 img 标签。

大家应该注意到了，笔者在描述这些元素时的用词区别。描述 wxml 元素使用的是“组件”，而描述 HTML 元素使用的是“标签”，这是符合规范称呼的。HTML 是标记语言，它的标签主要是用来标记页面骨架，标签的属性也比较少。但组件不同，组件除了标记的作用，它的属性一般也是非常丰富的。小程序官方文档中也将 view、text、image 称为组件，而并没有称为标签。

同 HTML 中的 img 标签一样，image 组件需要设置一个 src 属性，该属性指向一张图片的路径，用来显示该图片。我们在项目的根目录下新建一个名为 images 的目录，用来存放 Orange Can 的所有图片资源。在 images 目录下新建 avatar 目录，然后将一些适合做头像的图片拷贝到 avatar 目录中。

如何将图片放入到小程序的目录中？微信 Web 开发者工具无法通过 Ctrl+C、Ctrl+V 的功能复制粘贴图片，也没有提供导入图片的功能。我们需要在操作系统中打开项目的目录，并将图片拷贝到对应的 avatar 文件夹中，小程序会自动刷新目录，并在开发工具中显示这些图片。

以上操作完成后，我们的过程目录将变成如图 3-5 所示的结构。

现在在 image 组件中加入属性 src：

```
<image src="/images/avatar/avatar-1.png"></image>
```

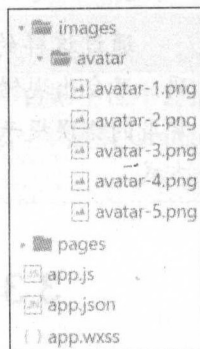


图 3-5 加入图片后的文件结构

当保存项目后，模拟器立刻会出现这张图片，但图片显示的高宽并不是图片本身的高宽。它被 MINA 框架设置成了宽度 300px、高度 225px，这也是小程序默认的图片高宽。如果我们不显示地指定图片高宽，所有图片都将保持这个默认值。

相对路径与绝对路径

在小程序中同样有相对路径和绝对路径的区别。上面我们在设置 image 组件的 src 属性时，使用的是绝对路径，它以“/”开头，“/”代表根目录。我们也可以使用相对路径来为 image 指定图片路径，比如，将代码中 image 组件的 src 属性改写为相对路径：

```
<image src="../../images/avatar/avatar-1.png"></image>
```

路径中的“..”表示向上一级。这里由于使用绝对路径更加简洁，所以我们使用绝对路径。

代码中间部分使用一个 view 组件包裹一个“开启小程序之旅”的 text 组件来实现按钮部分。由于还没有编写样式，所以暂时它还不能呈现为一个按钮的形状。目前整个页面看起来是这个样子的，如图 3-6 所示。

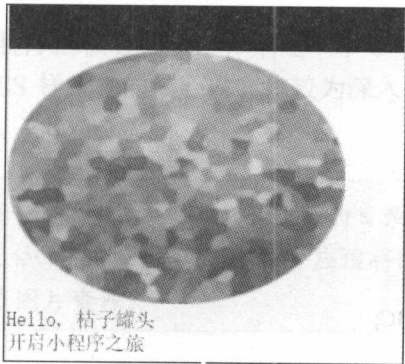


图 3-6 还没有加入样式代码时 welcome 页面的样子

现在来编写 welcome 页面的样式。小程序编写样式的语言就是 CSS，我们应该将 CSS 代码写在页面的 wxss 文件中。在编写 welcome.wxss 文件之前，首先在 welcome.wxml 文件中给每个需要样式的组件加入样式名称 class name。

代码清单 3-6 welcome.wxml

```
<view class="container">
  <image class="avatar" src="../../images/avatar/avatar-1.png"></image>
  <text class="motto">Hello, 桔子罐头</text>
  <view class="journey-container">
    <text class="journey">开启小程序之旅</text>
  </view>
</view>
```

就和我们在 HTML 中编写 CSS 名称一样，不是吗？

接着我们将下面这段 CSS 代码加入到 welcome.wxss 文件中。

代码清单 3-7 编写 welcome 页面的样式

welcome.wxss

```
.container{
  display: flex;
  flex-direction: column;
  align-items: center;
```

```

}

.avatar{
  width:200rpx;
  height:200rpx;
  margin-top:160rpx;
}

.motto{
  margin-top:100rpx;
  font-size:32rpx;
  font-weight: bold;
  color: #9F4311;
}

.journey-container{
  margin-top: 200rpx;
  border: 1px solid #EA5A3C;
  width: 200rpx;
  height: 80rpx;
  border-radius: 5px;
  text-align:center;
}

.journey{
  font-size:22rpx;
  font-weight: bold;
  line-height:80rpx;
  color: #EA5A3C;
}

```

让我们保存一下，看看页面发生了什么变化，如图 3-7 所示。



图 3-7 添加样式后的 welcome 页面

下面简单介绍一下这些 CSS 代码的作用。

- `.container` 是所有组件元素的容器样式。这里使用 Flex 布局的方式，来控制容器下子元素的排布规则。关于 Flex 将在下个小节里具体讲解。
- `.avatar` 设置头像图片的大小和位置。
- `.motto` 设置“Hello, 桔子罐头”这段文本的样式。
- `.journey-container` 设置了“开启小程序之旅”的外边框，使它们看起来更像一个按钮。`border-radius` 让这个外边框变成一个“圆角”的矩形。
- `.journey` 则设置了圆角矩形内的文本样式。

本书的主要目的是讲解小程序的核心知识，并不是一本 CSS 和 JavaScript 的基础语法书。限于书籍的篇幅，我们只对 CSS 样式中的核心内容作较为深入的讲解。下个小节，我们来学习小程序官方推荐的布局方式：Flex 布局。

真实项目中，图片资源尽量不要存储在小程序的目录中，因为小程序的大小不能超过 1MB，超过则无法真机运行和发布项目。应该将图片都存放在服务器上，让小程序通过网络来加载图片资源。

3.4 小程序所支持的 CSS 选择器

需要特别注意的是，小程序中的 CSS 只支持表 3-3 所示 6 种 CSS 选择器。

表 3-3 CSS 选择器描述

选择器	样例	样例描述
<code>.class</code>	<code>.sample</code>	选择所有拥有 <code>class="sample"</code> 的组件
<code>#id</code>	<code>#sample</code>	选择所有拥有 <code>id="sample"</code> 的组件
<code>element</code>	<code>view</code>	选择所有 <code>view</code> 组件
<code>Element,element</code>	<code>view,checkbox</code>	选择所有文档的 <code>view</code> 组件和所有的 <code>checkbox</code> 组件
<code>::after</code>	<code>view::after</code>	在 <code>view</code> 组件后边插入内容
<code>::before</code>	<code>view::before</code>	在 <code>view</code> 组件前边插入内容

虽然 CSS1、CSS2 和 CSS3 的选择器种类加起来总计几十种，但在小程序中只能支持以上几种。

同时需要注意的是，本地资源在 `wxss` 中是无法使用的。比如 `background-image`，如果使用的是本地图片，则无法显示，可以使用网络图片来代替本地图片。

3.5 Flex 布局

在 3.3 小节中，welcome.wxss 文件中的.container 样式使用了一个 display:flex 的样式。那么，什么是 Flex？

Flex 布局是 W3C 组织在 2009 年提出的一个新的布局方案，其宗旨是让页面的样式布局更加简单，并且可以很好地支持响应式布局。这并不是小程序所独有的技术，它本身是 CSS 语法的一部分。只不过早期时候，主流的浏览器对 Flex 布局的支持并不完善，造成了很多开发者不知道有这种布局的存在或者使用非常少，我们还是习惯使用传统的 position 和 float 属性来布局。但传统的布局方式有它的缺陷，比如像垂直居中就不是那么容易实现，Flex 可以很好地解决这些问题。

小程序能够非常好地支持 Flex 布局，并且这也是官方推荐的布局方式，我们结合代码清单 3-7 来学习一下 Flex 布局的基础知识。

Flex 也称为“弹性布局”，主要作用在容器上，比如代码清单 3-7 中样式为.container 的 view 组件，就是一个容器，它将页面中所有的元素都包裹起来。

我们使用 display:flex 将这个 view 变成了一个弹性盒子。设置 display:flex 是应用一切弹性布局属性的先决条件，如果不设置 display: flex，那么后续的其他相关弹性布局属性都将无效。

接着我们使用 flex-direction 这个属性指定 view 内元素的排列方向。这个属性可能的值有 4 个：

- row
- column
- row-reverse
- column-reverse

要理解这 4 个属性，首先要了解一个 Flex 布局非常重要的概念：轴。

我们知道，在一个平面直角坐标系里，轴有两个方向（就是 X、Y），分别是水平方向和垂直方向。一个弹性盒子，需要确定一个主轴。这个主轴到底是水平方向还是垂直方向，就由 flex-direction 这个属性的值来确定。如果 flex-direction 值为 row 或者 row-reverse，那么主轴的方向为水平方向，相反，如果值为 column 或者 column-reverse，那么主轴为垂直方向。选定主轴的方向后，另外一个方向的轴我们成为“交叉轴”。也就是说，主轴并不一定就是水平方向，交叉轴也并不一定就是垂直方向，主轴的方向由 flex-direction 的取值来决定。理解这一点尤其重要。我们来看图 3-8～图 3-11 所示。

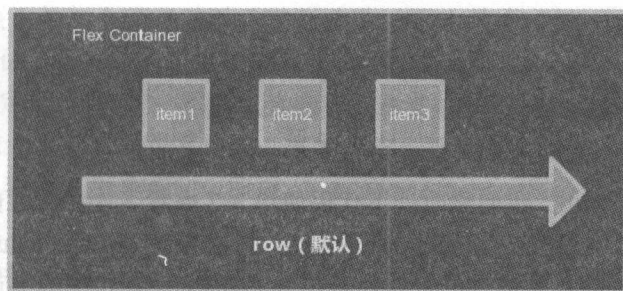


图 3-8 flex-direction:row 时的主轴方向

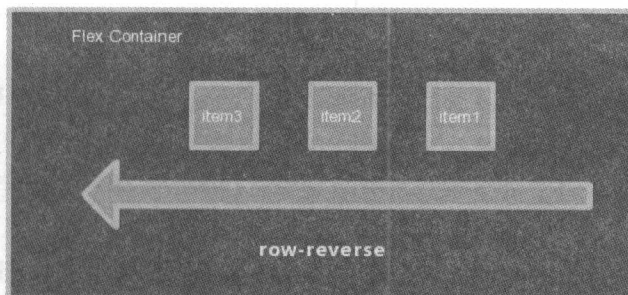


图 3-9 flex-direction:row-reverse 时主轴的方向及子元素排列

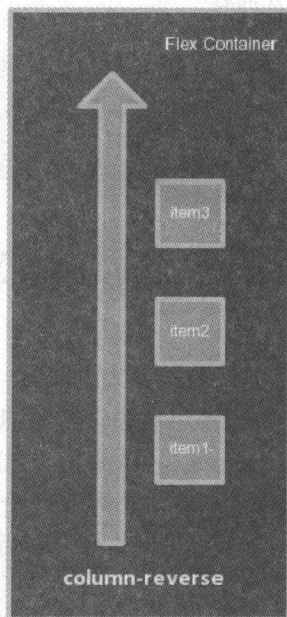
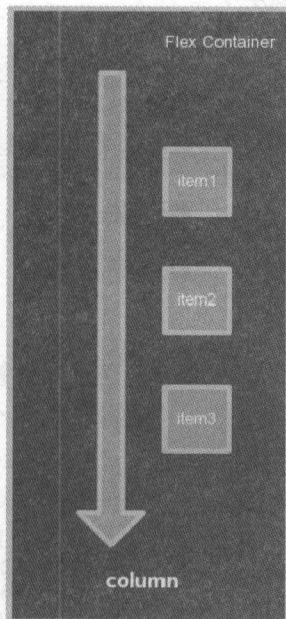


图 3-10 flex-direction:column 时主轴的方向及子元素排列

图 3-11 flex-direction:column-reverse 时主轴的方向及子元素排列

图 3-8 到图 3-11 显示了当 flex-direction 取不同值时，主轴方向及子元素排布的情况。注意观察每张图里 3 个小 item 的排布顺序，主轴方向不同，子元素排布的方向也不同。

- flex-direction:row 时, 主轴水平, 方向为自左向右
- flex-direction:row-reverse 时, 主轴水平, 但方向为自右向左
- flex-direction:column 时, 主轴垂直, 方向自上而下
- flex-direction:column-reverse 时, 主轴垂直, 方向自下而上

讲道理不如直接看效果。我们修改一下代码清单 3-7 来看下 4 种属性的效果。将代码清单 3-7 里 .container 样式中的 flex-direction 的值分别替换为 row、column、row-reverse、column-reverse。为了排除其他属性的干扰, 我们将 .container 样式中的 align-items:center 也先注释掉。注释代码的快捷键为 Ctrl+/. 得到的效果如图 3-12~图 3-15 所示。

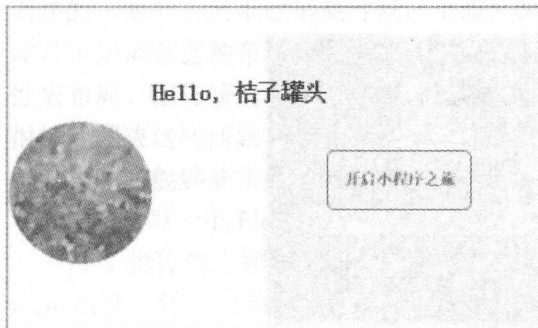


图 3-12 flex-direction 值为 row 时三元素自左向右

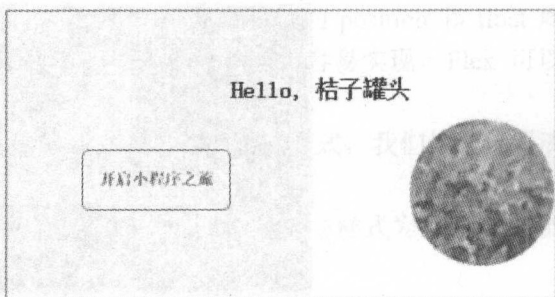


图 3-13 flex-direction 值为 row-reverse 时三元素自右向左



图 3-14 flex-direction 值为 column 时三元素自上而下



图 3-15 flex-direction 值为 column-reverse 时三元素自下而上

是不是非常清楚? 至于 row 和 row-reverse 这两张图中, “Hello, 桔子罐头” 往上偏移了一些, 是因为受到 welcome.wxss 样式表中其他 CSS 属性的影响。但这 3 个元素的主要排布方向特征正如我们预期的一样。

根据设计图的样式, 我们应该选择 flex-direction: column 作为我们的主轴。效果就如图 3-14 所显示的那样。

虽然 welcome 页面的 3 个元素已经呈现出了垂直排列，但他们还没有居中显示。`.container` 样式中的属性 `align-items: center`，可以让三元素水平居中。

`align-items` 属性定义子元素在交叉轴上如何对齐。这里特别要注意，`align-items` 定义的是“交叉轴”这个方向上子元素的对齐方式。比如，由于我们在 `.container` 样式中设置了垂直方向为主轴，那么交叉轴就是水平方向，所以 `align-items: center` 将设置三元素在水平方向上的对齐方式为居中。

当然，`align-items` 属性值不是只有 `center` 这一种，还有其他若干种取值。本书主要讲解小程序相关知识，限于篇幅这里就不再展开赘述这些 CSS 的相关知识。由于 Flex 布局在小程序里地位相当之高，本小节权当抛砖引玉，各位读者可以自行查找资料，更详细深入的学习 Flex 布局。

这里推荐一个学习方法。编程里的知识点是非常细小而琐碎的，学习不同的知识应该掌握不同的方法。对于学习 CSS 这类知识，笔者认为较好的学习方法应该是在实践中学习，而不是像我们上学时那样先把理论知识认真的学习一遍，甚至要求全部记住，这一点是不可取的。比如 Flex 布局的学习，我们首先应当大致浏览一下整个 Flex 的知识树，知道 Flex 解决了什么问题，有什么特点，大致有几类属性就够了。当我们在做项目遇到布局问题时，脑海里就能意识到 Flex 可能可以解决这个问题。接着我们抱着试试看的心态，带着目的去查找 Flex 布局的相关资料，即解决了问题，又能在实践中加深对 Flex 布局的理解，这比单纯死记硬背效果要好很多。人脑总是对形象化的东西记忆特别深刻，所以我们应当尽量在实践中学习知识。当然，也有可能 Flex 不能解决问题，但你查找和尝试解决问题的这个过程本身就是很好的学习手段。

3.6 小程序自适应单位 rpx 简介

不知道大家是否注意到，在 `welcome.wxss` 样式表中，我们绝大多数的长度单位都设置的是 `rpx` 这个全新的单位，比如 `margin-top: 100rpx`。在小程序里，长度单位既可以使用 `rpx`，也可以使用 `px`。使用 `rpx` 可以使组件自适应屏幕的高度和宽度，但使用 `px` 不会。要透彻地理解 `rpx` 需要对移动端分辨率有一定的了解，比如物理分辨率 `px`、逻辑分辨率 `pt` 等概念。这里只需要记住以下的结论，如果你不是很明白本章的内容，也没有关系，只需要记住结论即可，并不影响我们的开发。

建议以 iPhone 6 的宽度 750 个物理像素作为标准，来做设计图。在此宽度下，这张设计图里每个元素的尺寸转换到小程序样式时，转换比例为 1 物理像素 = 1rpx = 0.5px。rpx 和 px 就是小程序样式里可以使用的两种长度单位。

举个例子，我们的 welcome 页面设计图的宽度总长是 750 像素，它是以 iPhone 6 的尺寸来设计的，而其中的头像图片高宽为 200 像素×200 像素。如果想在 iPhone 6 里正确地显示这张 200 像素×200 像素的图片，那么相应地 `image` 组件的高宽应该设置为多少呢？

答案是要不就设置为高 200rpx，宽 200rpx，要不就设置为高 100px，宽 100px。这两个单位，在 iPhone 6 下显示效果一样，但如果我们将模拟器切换到其他机型，这两种不同的单位就会出现差异。rpx 将随着屏幕尺寸的变化而变化，但 px 不会。那么到底选择 rpx 还是选择 px 呢？这取决于你需要元素随着移动设备尺寸的变化而变化，还是让元素始终保持不变，需要具体问题具体分析。

对于 margin-top 或者是 image 组件的高宽，很多时候，需要他们随着设备的尺寸不同动态地变化，以保持页面元素之间的分布可以保持“一定的比例关系”，这种情况下应该使用 rpx。来看下面这个例子。

我们现在将模拟器的机型调到 iPhone 4，在 iPhone 4 机型下，welcome 页面呈现的 UI，如图 3-16 所示。

可以看到，虽然页面的整体“变小了”，但 3 个元素之间、元素与页边距之间的比例还是非常和谐和美观的。

接着我们将 image 组件的样式.avatar 更改为以下代码：

代码清单 3-8 将 image 组件的高宽单位由 rpx 更改为 px

```
.avatar{  
  width:200px;  
  height:200px;  
  margin-top:160rpx;  
}
```

页面将在模拟器 iPhone 4 机型下呈现如图 3-17 所示。

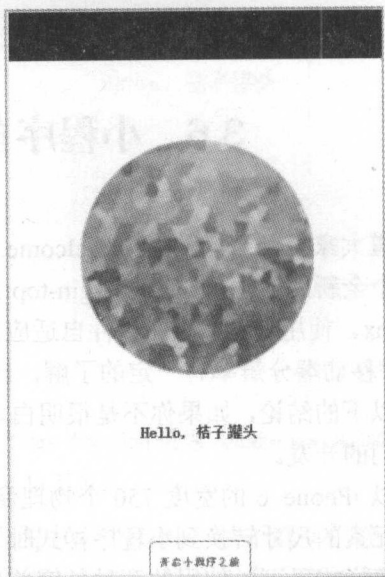


图 3-16 iPhone 4 下 welcome 页面的 UI 样式 图 3-17 将 image 图片高宽单位更改为 px 后的 UI 效果

将 image 组件的长度单位由 rpx 更改为 px 后，整个页面元素不再“协调”，image 变得很大”。这是因为 px 不会根据屏幕尺寸做自适应。

从这两张图可以看到，当 image 的高宽使用 rpx 做单位时，无论是在 iPhone 6 下，还是在 iPhone 4 下，整个页面都可以保持很好的布局，因为 rpx 会根据屏幕尺寸做出自动的调整。

开发者们可以自己将 welcome.wxss 里的 rpx 和 px 相互替换一下，或者多调整一下模拟器的机型，来感受一下 rpx 和 px 的不同。

那是不是 rpx 就是万能的，我们可以将页面里的所有元素的长度单位都换成 rpx 呢？来看看下面这个例子，在 welcome.wxss 里有一段这样的代码：

代码清单 3-9 welcome.wxss

```
.journey-container{
  margin-top: 200rpx;
  border: 1px solid #405f80;
  width: 200rpx;
  height: 80rpx;
  border-radius: 5px;
  text-align:center;
}
```

journey-container 设置了“开启小程序之旅”这段文本的外边框。为什么其他的元素我们都使用 rpx 为单位而唯独 border 这个属性使用的是 1px 呢？。因为我们讲过，rpx 是自适应单位，它通常非常适合用来控制图片的高宽和元素之间的间距（通常这些元素需要随着屏幕尺寸的不同而动态变化）。但我们考虑一下，border 这个属性需要动态变化吗？不需要。如果 border 动态变化，那么它会在屏幕尺寸较大的手机上变得很粗，这并不是我们想要的效果。所以这里应当将 border 的单位设置为 px。同理，使用 px 作为 border-radius 的单位。

最后，我们为什么要强调最好是在 iPhone 6 的尺寸下做设计图呢？因为只有 iPhone 6 的尺寸下，设计图里的 1 个像素才满足下面的转换关系：

$$1 \text{ 物理像素} = 1\text{rpx} = 0.5\text{px}$$

如果不以 iPhone 6 的标准来做设计图，也是可以的。但非 iPhone 6 的尺寸下，设计图与 rpx、px 的转换关系就不是整数倍的，计算起来比较麻烦，所以建议设计图最好以 iPhone 6 的尺寸标准来设计，这样换算起来很方便。这也是官方建议的一个设计标准。

如果我们足够细心，可以看到小程序的模拟器选择项下，给出了每种机型的分辨率。要强调的是，这里的分辨率指的是逻辑分辨率 pt，而非物理分辨率。以 iPhone 6 为例，模拟器里给出的分辨率是：375 × 667；Dpr: 2

它的意思是：iPhone 6 的水平方向有 375 个逻辑像素点，而竖直方向有 667 个逻辑像素点，每个逻辑像素点包含 2 个物理像素点。开发者一定要注意逻辑像素和物理像素的区别。我们通常在 PS 里做的设计图，它的像素可以简单理解为物理像素。

再次提醒开发者，1 物理像素不等于 1px。假设有一张图片在操作系统下显示宽度为 750 个像素，我们现在想让这个图片水平方向充满整个页面。如果你直接在页面里（iPhone 6 模拟机型下）将图片宽度设置为 750px，这是不对的。正确的设置方法为 750rpx 或者 375px，才能让图片水平填满小程序。因为，iPhone 6 下：

1 物理像素=1rpx=0.5px。

3.7 全局样式文件 app.wxss

到目前为止，welcome 欢迎页面和实际图相比还有一些不一样的地方，比如字体。设计图里的字体使用的是微软雅黑，而目前的字体还是小程序默认的字体。

最简单的更改字体的方法是在 wxss 页面中加入如下代码：

代码清单 3-10 改变页面 text 组件字体

welcome.wxss

```
text{
  font-family: Microsoft YaHei;
}
```

代码会将 welcome 页面中的所有 text 组件的字体更改为微软雅黑。那我们思考一个问题，假如现在有 100 个页面，而 100 个页面里几乎所有的字体都应该是微软雅黑。在 100 个页面里重复设置字体这并不是一个很好的解决方案。

所以，我们需要有一个全局样式表，可以为所有页面设置“默认”样式。小程序为我们提供了一个这样的样式表文件，就是前面提到过的 app.wxss 文件。

我们将代码清单 3-10 的代码，复制到项目根目录下的 app.wxss 文件中。虽然这段代码不在 welcome.wxss 页面样式表中，但依然可以使 welcome 页面的 text 组件字体更改为“微软雅黑”。还可以在这里设置一些其他的公共样式，比如字体大小 font-size、字体颜色 color 等。

如果不想在某个页面中使用全局默认样式，那么只需要在相应页面的 wxss 文件中重新定义这个样式即可。小程序会优先选择页面的 wxss 文件，而不是 app.wxss 文件。

3.8 页面的根元素 page

到目前为止，我们的 welcome 页面已经像那么回事儿了。但页面的样式和设计图还不太一样，设计图中整个页面呈现的是橘红色，而现在的页面还是白色。那么，来修改一下页面的背景颜色吧。

要修改页面整体的背景色，需要寻找一个包裹所有页面元素的容器，并设置这个容器的背景色。那么，首先尝试给页面最外层 view（class = "container" 的这个 view）一个背景色。在 welcome.wxss 文件中的.container 样式里新增属性 background-color: #ECC0A8。



代码清单 3-11 向.container 中新增背景色

welcome.wxss

```
.container{
  display: flex;
  flex-direction:column;
  align-items: center;
  background-color:#ECC0A8;
}
```

接着保存预览一下增加样式后的页面，它将呈现如图 3-18 所示的效果。

并不是整个页面都呈现出橘红色，只是有元素占据的地方才呈现出橘红色。原因是因为最外层的 container view 没有固定的高度，它的高度由其内部子元素决定，所以橘红色部分的下边刚好和按钮的下边重合。如何解决这个问题呢？

可以通过给 container view 一个固定的高度来解决这个问题，但这并不是最好的办法。因为在不同的机型上，屏幕的尺寸是不一样的，固定的高度无法去适配不同的机型，可能出现滚动条，也可能橘红色无法覆盖整个页面。

当然，用我们前面学到的 rpx 是可以解决这个问题的，将 container view 的高度单位设置为 rpx，就可以让它随着不同的机型进行自适应调整。但这看起来很蠢，高度具体设置多少，还需要我们了解 iPhone 6 的屏幕分辨率。所以，这依然不是一个很好的解决方案。

其实，在 container view 的外边，小程序还有一个默认的容器元素：page。我们可以在开发工具中的 Wxml 这个 Pannel 中看一下 welcome 页面的页面结构，如图 3-19 所示。



图 3-18 设置容器 view 后的页面效果

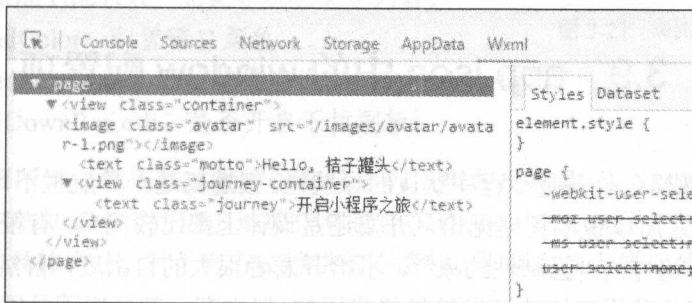


图 3-19 用 Wxml Pannel 查看页面结构

在 class="container" 这个 view 的外边还有一个容器元素：page。这是 MINA 框架为大家默认添加的。每个小程序页面的最外层都有这个 page 元素。page 元素代表着页面这个整体，所以只需要对 page 设置背景色即可实现设计图里的效果。在 welcome.wxss 文件的尾部追加以下样式：

```
page{  
  background-color:#ECC0A8;  
}
```

保存后，页面将呈现出如图 3-20 所示的效果。

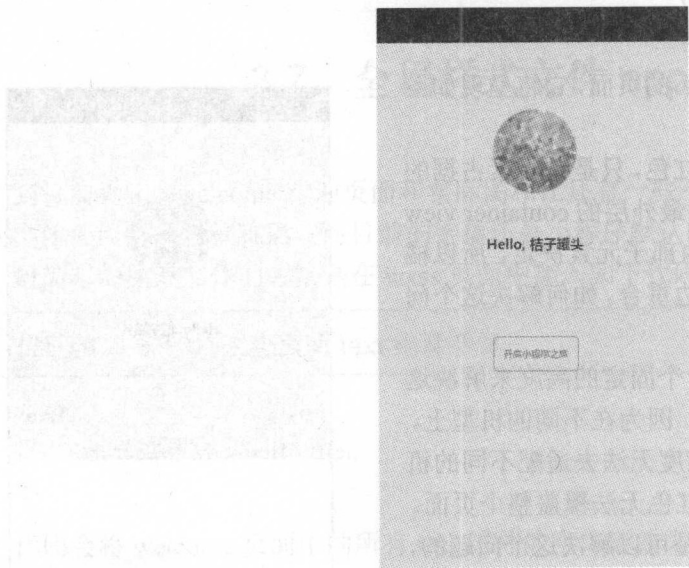


图 3-20 对 page 设置背景色后的页面效果

page 代表着整个页面的容器，如果想对页面整体做样式或者属性设置，那么应该考虑 page 这个页面的根元素。

现在，welcome 页面的顶部还有一块儿黑黢黢的长条，这实在是太难看了。我们下个小节来解决这个问题。

3.9 app.json 中的 window 配置项

图 3-20 的顶部黑色长条是小程序默认的导航栏。很遗憾这个导航栏不可以隐藏或者取消，它必须存在。我们之所以说小程序无论从开发还是设计上都比较简单，有很大一部分原因是因为小程序做了很多这样的“强制性约束”，不给开发者很大的自由度，自然简单。这固然是因为小程序设计初衷就是用来快速开发轻量级应用的，但也有一部分原因是因为小程序目前还处于生态的早期阶段，官方没有那么多精力支持非常丰富的组件接口。

既然这个导航栏无法取消，如何让整个页面只有一种颜色呢？下面我们考虑将导航栏的颜色和页面的背景色设置成同一个颜色。

还记不记得在 3.2 小节中，我们使用了 app.json 的一个配置项 pages，用来注册小程序页面文件？这一小节，我们来学习 app.json 的另外一个配置项 window。

window 配置项用来设置小程序的状态栏、导航栏、标题和窗口的背景色。我们先来学习 window 配置项下能够更改导航栏颜色的属性：navigationBarBackgroundColor。在 app.json 文件中加入一个 window 配置项，并设置 window 的属性 navigationBarBackgroundColor 值为 #ECC0A8。更改后的 app.json 文件代码如下：

代码清单 3-13 设置导航栏的背景色

app.json

```
{
  "pages": [
    "pages/welcome/welcome"
  ],
  "window": {
    "navigationBarBackgroundColor": "#ECC0A8"
  }
}
```

保存后，我们发现 welcome 页面的样式已经变成如图 3-21 所示。

小程序顶部的导航栏已经被“隐藏”了。当然这不是真的被隐藏，导航栏依然存在于小程序中，只不过我们通过设置导航栏和页面的颜色模拟了这种效果。

现在，我们的欢迎页面是不是很像各类 iOS 和 Android App 的启动页面？

当然，window 这个选项下并不是只有 navigationBarBackgroundColor 这一个属性，它还有以下几种属性：

- navigationBarTextStyle 配置导航栏文字颜色，只支持 black/white。
- navigationBarTitleText 配置导航栏文字内容。
- backgroundColor 配置窗口颜色。
- backgroundTextStyle 下拉背景字体，仅支持 dark/light。
- enablePullDownRefresh 是否开启下拉刷新。



图 3-21 welcome 页面的最终样式

正如本书前面所讲，把这些文档内容在这里列出来并没有意义，这些都是官方文档里的内容。本书会将这些知识点尽可能多地编排和融合在 Orange Can 项目中，在案例实践中演示这些组件、属性的具体用法。对于官方文档讲的不清楚、不明白或者错误的地方，本书也会做出补充和修正。

读者朋友应该关注本书章节的标题，每个标题代表着小程序的一个重要知识点，整本书的标题将构成小程序的知识体系框架。但每个知识点下有很多的属性，本书并不会一一列举，因为这些属性的使用方式都可以通过本书的案例并结合官方的文档一目了然。

对于每个小程序的知识点，我们在学习阶段最需要关心的只有两点——有什么用和怎么用。比如 window 这个配置项，我们只需要关心以下两点即可：

- window 是做什么的？
- 怎么使用 window 这个配置项？

至于 window 下面的属性值，建议具体问题具体对待，不需要现在就搞明白。把这些属性值放到实际的工作项目中学习，不仅节约时间而且印象更加深刻。比如，我们这里需要用到 `navigationBarBackgroundColor`，把这个属性配置到项目后，就知道 `navigationBarBackgroundColor` 的意义了。

Orange Can 项目的后期代码里，我们还会使用到 window 的其他配置项，让我们继续项目，逐步学习。

书籍绝大多数是用来引导入门和分享思想的，它不应该替代官方的 API 文档。API 文档不同于“tutorial”或者“get started”（优秀的开发文档都有这两个部分），它一般用于查阅，是工具而非教程，通常都非常简洁。这里分享一个学习 API 的方法，就是“试”。

对于 window 这个配置项，你只需要将其他几个属性加入到 window 中，再更改几个可能的属性值，就可以立刻即时地预览到属性值的效果。如果属性值的效果不符合你的预期，就具体去分析为什么会出现这种情况。不知不觉中，你对整个 API 就会越来越熟悉。

当然，还是我们之前提到过的，不需要把整个 API 文档都试一遍，当你需要解决问题时，结合具体的案例再来“试”这些 API。

第 4 章

文章列表页面

在编写完“welcome”欢迎页面后，我们将编写文章列表部分。文章列表部分展示了一个 banner 轮播图与一组文章。在编写此部分功能时，我们会介绍使用 swiper 组件构建 banner 轮播图以及 swiper 组件的其他属性；详细介绍 image 组件的 4 种缩放模式与 9 种裁剪模式，全面理清官方文档中没有详细说明的一些知识点。

除此之外，小程序最重要的数据绑定将出现在本章中。数据绑定是小程序中最重要的概念，它是和传统的 Web 网页编程相比最大的不同。在小程序中，几乎所有和数据相关的操作都只能使用数据绑定来完成。

本章我们还会重温在传统网页中经常使用到的“事件”，一起来看看小程序中的事件和传统网页中的事件有什么异同。

4.1 文章列表页面元素分析及准备工作

上一章我们一起完成了 Orange Can 的第一个页面：welcome 欢迎页。虽然简单，但它基本描述了小程序的开发模式。本章我们来构建第二页面：文章页面。该页面的设计图可参考本书彩页部分图片。

文章页面主体部分由两部分构成，上半部分是一个轮播图，下半部分是文章列表。轮播图目前来说已经成为各大电商网站首页的标配元素，如图 4-1 所示。



图 4-1 京东首页的 banner 轮播图

轮播图每隔几秒钟图片会自动更换一张。在小程序中，我们不需要自己编写代码来实现这样的轮播图效果，小程序已经提供了一个现成的组件——swiper。

下半部分是多篇文章构成的文章列表。每篇文章包含文章标题、文章头图、文章概要、评论数和阅读数。我们依然只需要使用在上一章中介绍的 3 个组件：view、image 和 text，即可实现这个文章列表。

先来创建文章列表页面的相关文件。

在 pages 目录下新建一个名为 post 的目录，然后依次在 post 目录下新建阅读页面所需要的 4 个文件 post.wxml、post.wxss、post.json 和 post.wxss。

建议使用 3.2 小节中介绍的快捷创建页面目录及文件的方法，一个个地创建 4 个页面文件实在太麻烦。如果你是一个个地手动新建页面文件，那么请注意添加 3.2 小节中我们讲到的页面默认必备代码，否则小程序会报错。创建后的项目目录如图 4-2 所示。

现在有个问题，我们要编写阅读页面，但我们的启动页面已经设置成了 welcome 欢迎页面。在不编写“开启小程序之旅”这个 button 跳转页面功能之前，我们没办法看到文章页面。实现 button 跳转页面的功能，需要用到小程序事件和 JavaScript 代码，我们先尽可能地熟悉小程序页面骨架的编写，稍微复杂一些的事件和 JavaScript 代码留在后面的章节讲解。



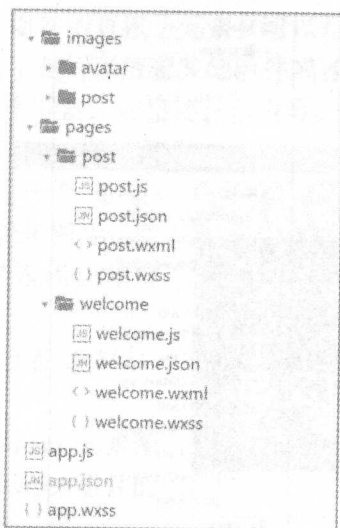


图 4-2 加入阅读页面后的目录结构

先做一个调整。在之前章节里我们提到过，小程序启动后显示的首页，由 `app.json` 文件里 `pages` 数组的第一个元素决定。我们暂时先将首页调整成 `post` 页面。

在 `app.json` 的 `pages` 数组里加入 `post` 页面路径，注意它必须是 `pages` 数组的第一个元素，位于 `welcome` 页面之前。代码清单如下：

代码清单 4-1 注册 `post` 文章页面

app.json

```
{
  "pages": [
    "pages/post/post",
    "pages/welcome/welcome"
  ],
  "window": {
    "navigationBarBackgroundColor": "#ECC0A8"
  }
}
```

更改完成后，现在保存或者重新编译项目，启动页面将不再是 `welcome` 页面，而变成了 `post` 页面。

在小程序的 `images` 目录下新建一个子目录 `post`，并将阅读页面所需要的图片素材拷贝到该目录下。读者可以自行选择自己喜欢的图片素材，或者访问前言中【项目资源文件及源代码及如何阅读本书】中所提供的地址来获取素材。图片的像素大小要大于或者等于 750（宽）和 600（高），过小的图片会出现“留白”的情况。图 4-3 显示了本书所使用图片素材的目录情况，本书后续将不再一一展示图片目录，请读者朋友自行将图片路径填写正确。

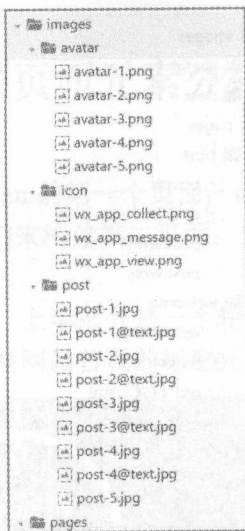


图 4-3 素材所在目录

完成以上准备工作后，就可以开始编写文章页面了。

4.2 swiper 组件

本小节，我们来学习小程序提供的滑动视图容器——swiper 组件，在 post.wxml 中加入以下代码：

代码清单 4-2 加入 swiper 组件

post.wxml

```

<view>
  <swiper>
    <swiper-item>
      <image src="/images/post/post-1@text.jpg/wx.png" />
    </swiper-item>
    <swiper-item>
      <image src="/images/post/post-2@text.jpg/wx.png" />
    </swiper-item>
    <swiper-item>
      <image src="/images/post/post-3@text.jpg/wx.png" />
    </swiper-item>
  </swiper>
</view>
  
```

最外层的<view></view>将作为整个页面的容器，在 view 的内部，我们加入了一个 swiper 组件。swiper 组件主要由多个 swiper-item 组成，可以定义任意多个 swiper-item。



同时，需要注意的是，swiper 组件的直接子元素只可以是 swiper-item，如果放置其他组件，则会被自动删除。但 swiper-item 下是可以放置其他组件或者元素的。

swiper-item 元素仅仅只是一个容器，如果要显示内容，需要在 swiper-item 容器下再添加元素内容。如代码清单 4-2 所显示的一样，我们在每个 swiper-item 内都加入了一个 image 组件，用来显示 UI 效果图中的轮播图片，图片路径请根据自己项目的实际情况做出相应的修改。

添加完代码后，保存一下项目看看结果，如图 4-4 所示。

swiper 组件的第一个 swiper-item 元素图片已经显示出来了。在动手设置 swiper 组件的样式前，首先在 post.wxss 文件内，将 swiper 组件的高度和高度设置好。

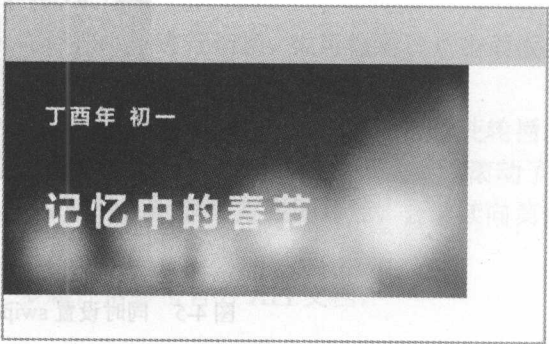


图 4-4 添加 swiper 组件后的 UI 效果

代码清单 4-3 设置 swiper 组件的样式

post.wxss

```
swiper{
  width:100%;
  height:600rpx;
}
```

添加完代码后，保存预览，发现图片的显示尺寸依然不正确。宽度没有呈现 100%，高度也不是期望的 600rpx。还需要对 image 组件设置同样的样式，在 post.wxss 中添加 image 组件的样式，添加完成后的页面代码如代码清单 4-4。

代码清单 4-4 添加 image 组件的样式

post.wxss

```
swiper{
  width:100%;
  height:600rpx;
}

swiper image{
  width:100%;
  height:600rpx;
}
```

此时再次预览小程序，发现样式已经符合预期的效果了，如图 4-5 所示。

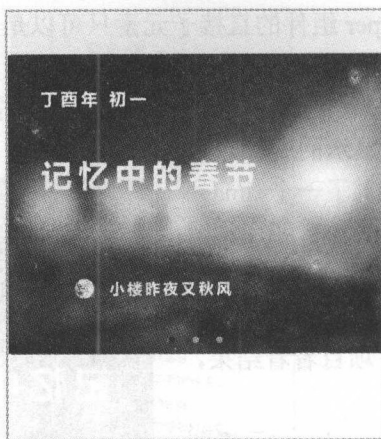


图 4-5 同时设置 swiper 和 image 样式后的效果

这里需要同时设置 swiper 组件和 image 组件的高宽，才能达到预期的效果。如果只设置 image 组件的高度同样是不可以的，读者可以把 swiper 组件的样式注释掉，看看只对 image 设置样式的结果。

要实现轮播效果，还要为 swiper 组件添加一些属性，分别是：indicator-dots、autoplay、interval，如代码清单 4-5 所示。

代码清单 4-5 向 swiper 中添加一些属性

post.wxml

```
<view>
  <swiper indicator-dots="true" autoplay="true" interval="5000">
    <swiper-item>
      <image src="/images/post/post-1@text.jpg" />
    </swiper-item>
    <swiper-item>
      <image src="/images/post/post-2@text.jpg" />
    </swiper-item>
    <swiper-item>
      <image src="/images/post/post-3@text.jpg" />
    </swiper-item>
  </swiper>
</view>
```

保存后预览一下效果。图片开始了轮播，每隔 5 秒钟更换一张。同时 swiper 组件上出现了 3 个小圆点，用来指示当前图片。

简单介绍一下这 3 个属性。

- indicator-dots

Boolean 类型。用来指示是否显示面板指示点（上文提到的 3 个小圆点就是面板指示点，默认为 false）。

- `autoplay`
Boolean 类型。用来决定是否自动播放，默认为 `false`。
- `interval`
Number 类型。用来设置 `swiper-item` 的切换时间间隔，默认为 5000 毫秒。

除了自动切换图片，`swiper` 组件还可以通过拖动图片来进行切换，也可以通过点击面板指示点来切换。

官方在 0.11.12210 版本中为 `swiper` 组件新增了一个 `circular` 属性，这个属性可以使轮播图循环滚动。如果 `circular` 为 `false`，那么当 `swiper` 组件滚动到第三张图片后就无法继续滚动了；但如果增加一个 `circular` 为 `true` 的属性，则当 `swiper` 组件滚动到第三张图片后，会继续向第一张图片滚动，从而形成循环滚动。

`swiper` 组件的属性使用方式都比较简单，更多属性请参考官方 API 文档。

4.3 Boolean 值的陷阱

这里介绍一个文档里没有提到的属性：`vertical`。这个属性将指明 `swiper` 组件面板指示点的排布方向是水平还是垂直，将 `vertical = "true"`，加入到 `swiper` 的属性中。保存后，我们发现 `swiper` 组件的面板指示点由原来的水平排布更改为垂直排布，出现在组件的右侧。

那如果把 `vertical` 属性改为 `false` 呢？形如，`vertical = "false"`。此时，面板指示点如何排列？它依然和 `vertical = "true"` 时的排列方向一样，呈垂直排布。为什么会出现这样的情况？我们可以把 `vertical` 的属性值更改为任何字符串，再看看效果。形如 `vertical="aaa"`、`vertical="bbb"` 等属性值都会让指示面板呈垂直分布。而 `vertical=""` 则呈水平分布。

我们应该可以从上面的属性举例中找出原因了。即使我们将 `vertical` 的值设置为 `false`，但这里的 `false` 并不是 Boolean 类型，而是一个字符串。只要不是空字符串，那么在 JavaScript 里都会认为这是一个 `true`。所以，设置 `vertical="false"`、`vertical="aaa"` 和 `vertical="bbb"`，效果是一样的：`vertical` 属性被认为设置成了 `true`。

如果想让面板指示点水平排列，有以下几种方式：

- 不加入 `vertical` 属性
- `vertical = ""`
- `vertical = "{{false}}"`

以上几种写法，小程序都会认为你将 `vertical` 属性设置成了 `false`。第三种写法，是我们后面要学习到的核心知识：数据绑定。这种写法，让 `{{false}}` 里的 `false` 被认为是一个 Boolean 类型的变量，而不是一个字符串，从而实现 `false` 即是假，`true` 即是真的效果。

当然，`swiper` 的 `vertical` 属性如果设置错误，一眼就能看出问题来。但如果是其他无法直接在 UI 上表现的属性出现了真假错误，就不是那么容易排查了，可能会浪费掉我们大量的时间。所有组件的 Boolean 类型属性都有这样的 Boolean 陷阱，比如，本例中的 `indicator-dots` 和 `autoplay` 也存在这个问题。

4.4 构建文章列表的骨架和样式

完成了 swiper 轮播图后,我们继续来构建设计图中的下半部分——文章列表。设计图见本书彩页部分。

正如前文所讲,构建文章列表依然只需要我们熟悉 3 个组件: view、text 和 image。将代码清单 4-6 的代码添加在 swiper 组件代码后面。

代码清单 4-6 加入文章列表代码

post.wxml

```
<view class="post-container">
  <view class="post-author-date">
    <image src="/images/avatar/avatar-5.png" />
    <text>Jan 28 2017</text>
  </view>
  <text class="post-title">小时候的冰棍儿与雪糕</text>
  <image class="post-image" src="/images/post/post-4.jpg" />
  <text class="post-content">冰棍与雪糕绝对不是同一个东西。3 到 5 毛钱的雪糕犹如现在的哈根达斯, 而 5 分 1 毛的冰棍儿就像现在的老冰棒。时过境迁, ...
  </text>
  <view class="post-like">
    <image src="/images/icon/wx_app_collect.png" />
    <text>108</text>
    <image src="/images/icon/wx_app_view.png"></image>
    <text>92</text>
    <image src="/images/icon/wx_app_message.png"></image>
    <text>7</text>
  </view>
</view>
```

保存后,效果如图 4-6 所示。

由于还没有加入 CSS 代码,所以整个页面的布局乱七八糟,但文章列表所有的元素都已经呈现在了页面中。将代码清单 4-7 的代码加入到 post.wxss 文件中。

代码清单 4-7 添加文章列表代码

post.wxss

```
swiper{
  width:100%;
  height:600rpx;
}

swiper image{
```



```

width:100%;
height:600rpx;
}

.post-container{
  flex-direction:column;
  display:flex;
  margin:20rpx 0 40rpx;
  background-color:#fff;
  border-bottom: 1px solid #ededed;
  border-top: 1px solid #ededed;
  padding-bottom: 5px;
}

```

```

.post-author-date{
  margin: 10rpx 0 20rpx 10px;
  display:flex;
  flex-direction: row;
  align-items: center;
}

```

```

.post-author-date image{
  width:60rpx;
  height:60rpx;
}

```

```

.post-author-date text{
  margin-left: 20px;
}

```

```

.post-image{
  width:100%;
  height:340rpx;
  margin-bottom: 15px;
}

```

```

.post-date{
  font-size:26rpx;
  margin-bottom: 10px;
}

```

```

.post-title{
  font-size:16px;
  font-weight: 600;
  color:#333;
  margin-bottom: 10px;
  margin-left: 10px;
}

```

图 4-6 展示样式的 post 页面的效果

```
}
.post-content{
  color:#666;
  font-size:26rpx;
  margin-bottom:20rpx;
  margin-left: 20rpx;
  letter-spacing:2rpx;
  line-height: 40rpx;
}
.post-like{
  display:flex;
  flex-direction: row;
  font-size:13px;
  line-height: 16px;
  margin-left: 10px;
  align-items: center;
}
.post-like image{
  height:16px;
  width:16px;
  margin-right: 8px;
}
.post-like text{
  margin-right: 20px;
}
```

保存预览一下，效果将如图 4-7 所示。

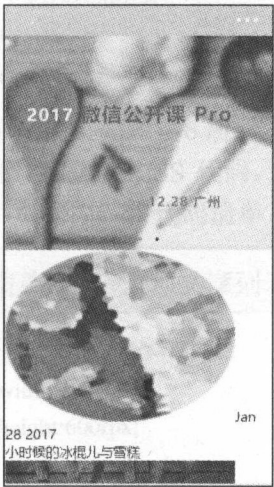


图 4-6 缺少样式时 post 页面的效果



图 4-7 加入样式后的文章列表

还有些小小的问题：“Jan 28 2017”和“108、92”这几个文本的字体大小与颜色都不太好。我们可以将一些默认的字体样式放在 app.wxss 全局样式表里。

代码清单 4-8 在全局样式表里加入默认字体样式

app.wxss

```
text{
  font-size:24rpx;
  font-family:Microsoft YaHei;
  color: #666;
}
```

保存后，日期和数量都呈现出 app.wxss 里设置的样式。

4.5 image 组件的 4 种缩放模式与 9 种裁剪模式

4 种缩放模式和 9 种裁剪模式如果从理论上完全精确理解，还是有稍许的难度的。但这里笔者建议各位开发者，没有必要完全从理论上搞清楚这些模式。当遇到具体问题时，尝试多去更换几个属性，找到最适合自己需求的属性即可。

来看看上个小节中雪糕图片的显示问题，很明显整个图片被压缩变形了。这并不是我们想要的结果。

post-image 这个元素的高宽分别被设置成 340rpx 和 100%（iPhone 6 下就是 750rpx），而雪糕图片素材原始高宽分别为 600px 和 750px。

在现实的项目中，我们经常要面对原始图片的尺寸和设计图里的尺寸不一样的情况（尤其是原始图片高宽是未知和不固定的情况，比如动态从网络获取图片）。在这种情况下，我们必须要有有所舍弃，或放弃等比例，或裁剪掉图片的一部分。接受不完美，也是编程中很重要的心态，如何选择，需要看业务上的需求。具体到文章列表图片，我们需要的是保持宽高比，接受部分裁剪（现实项目中，绝大多数情况下，图片保持比例、允许裁切是最普遍的需求）。

小程序的 image 组件提供了 4 种缩放模式和 9 种裁剪模式，来支持我们的选择。

4 种缩放模式

- **scaleToFill** 不保持纵横比缩放图片，使图片的宽高完全拉伸至填满 image 元素。
- **aspectFit** 保持纵横比缩放图片，使图片的长边能完全显示出来。也就是说，可以完整地显示图片。
- **aspectFill** 保持纵横比缩放图片，只保证图片的短边能完全显示出来。也就是说，图片通常只在水平或垂直方向是完整的，另一个方向将会发生截取。
- **widthFix** 宽度不变，高度自动变化，保持原图宽高比不变（0.11.122100 版本新增）。

9 种裁剪模式

- top 不缩放图片，只显示图片的顶部区域。
- bottom 不缩放图片，只显示图片的底部区域。
- center 不缩放图片，只显示图片的中间区域。
- left 不缩放图片，只显示图片的左边区域。
- right 不缩放图片，只显示图片的右边区域。
- top left 不缩放图片，只显示图片的左上边区域。
- top right 不缩放图片，只显示图片的右上边区域。
- bottom left 不缩放图片，只显示图片的左下边区域。
- bottom right 不缩放图片，只显示图片的右下边区域。

每种模式的字面意思都很好理解。要更改图片的裁剪或缩放模式，只需要给 image 组件加上一个 mode 属性值。

我们来看一下这 13 种不同属性值的实际效果。

4.5.1 scaleToFill

给代码清单 4-6 中 class = "post-image" 的 image 组件加上一个 mode 属性值。如代码清单 4-9 所示。

代码清单 4-9 image 组件的 mode 属性值

post.wxml

```
<image class="post-image" src="/images/post/post-4.jpg"
mode="scaleToFill" />
```

保存预览一下，文章图片好像并没有发生任何变化。

这是因为 scaleToFill 模式是缩放的默认模式，如果缺省了 mode，则小程序也会以 scaleToFill 的模式来缩放图片。scaleToFill 模式将改变图片的高宽比，强行让图片更改为样式指定的尺寸，使图片变形（如果原始图片的宽高比例和要缩放的目标宽高比例相同，则不会变形，只是整体上放大或者缩小了）。

4.5.2 aspectFit

再看看 mode=aspectFit 的情况，如图 4-8 所示。

同样不是我们要的效果。官方文档的解释：aspectFit 模式保持纵横比缩放图片，使图片的长边能完全显示出来。这个解释从字面上来看，并不是很容易理解，我们可以这样理解这种模式。

假想有一个容器，这个容器的高宽等同于 image 将要被缩放的目标尺寸。比如在当前的事例中，这个容器的高宽就是样式 post-image 所设置的高 320rpx，宽 100%（iPhone 6 下为 750rpx）。aspectFit 的特点就是保持图片



图 4-8 mode=aspectFit 的效果



不变形，且容器要“刚好”将这个图片装进去。注意，是“刚好”。如果原始图片比容器大，就要被等比例缩小；而原始图片如果比容器小，则要被等比例放大。一直放大或者缩小到图片的某一条边刚好和容器的一条边重合，而另一条边不能超出容器，也不能小于容器太多。

再回头看看图 4-8，宽刚好和容器相贴合，而高则刚好能被容器装进去，既没有超出容器，也没有比容器矮太多。同时整个图片保持了原始图片的宽高比，没有变形。

所以官方文档的解释，没有把这种模式的特点完全描述出来。

事例里用到的图片是大于容器的，所以图片会被缩小；开发者们可以尝试着用一张小于容器的图片替换这张雪糕图，试试 `aspectFit` 的效果。

效果如图 4-9 和图 4-10 所示。

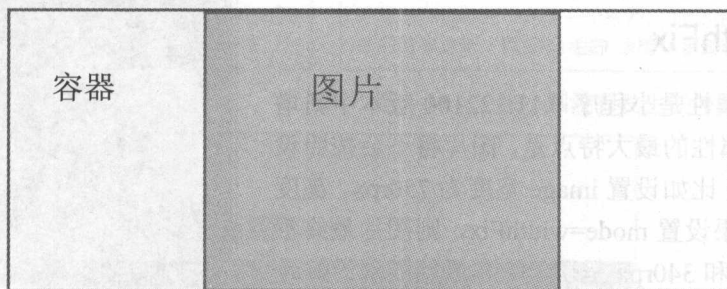


图 4-9 aspectFit 效果，“刚好”贴合容器

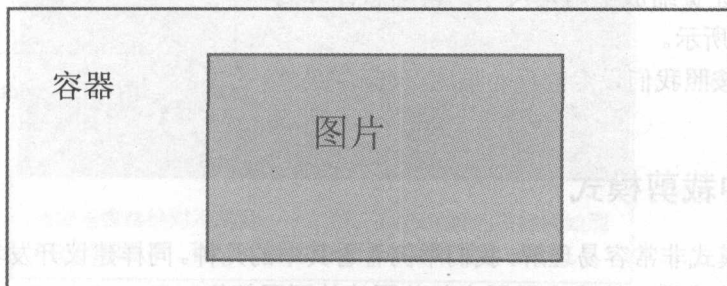


图 4-10 aspectFit 不可能出现的情况，图片没有同容器贴合

4.5.3 aspectFill

再看看 `mode=aspectFill` 的效果，如图 4-11 所示。

`aspectFill` 同样保持图片的高宽比不会变形。但它有个特点，它会让图片完全填满整个容器，类似于 `scaleToFill` 这种模式。不同的是，`scaleToFill` 会改变图片的高宽比，而 `aspectFill` 不会。

用我们上面提到的“容器”的观点来理解 `aspectFill`。既然 `aspectFill` 一定要填满整个容器，那么首先要让这张图片的整体尺寸是大于这个容器的，不能留下任何的空白。对于原始尺寸小于容器的，就等比例放大图片（任意一边小于容器都需要放大，否则就会留下空白），让



图 4-11 mode=aspectFill 的效果

图片的某一边刚好接触容器的一边，同时另外一边又不会小于容器（可以超出，因为这一边会被截取）。

如果原始尺寸大于容器，则需要按比例缩小，缩小的要求同样是一边刚好接触容器，另外一边要等于或者超出容器。这样就保证了图片可以完全填满整个容器，但某一边会发生截取。那么问题来了，如何截取？在超出容器的这一边上，是保留图片的上部、中部还是下部？答案是：中部。

注意观察图中的雪糕和原始图片，发现正中间部分被保留了下来。开发者可以自行多换几张素材图看看截取的效果。

4.5.4 widthFix

widthFix 属性是小程序 0.11.122100 版本中新增的属性。这个属性的最大特点是，图片将不会按照设定的尺寸呈现，比如设置 image 宽度为 750rpx，高度为 340rpx，如果设置 mode=widthFix，则图片最终不会按照 750rpx 和 340rpx 呈现，除非原始图片恰好是这个尺寸。

这个属性让宽缩放至指定尺寸，再动态计算高度，如图 4-12 所示。

虽然宽度按照我们设定的尺寸呈现，但高度突破了 340rpx。



图 4-12 widthFix 的效果

4.5.5 9 种裁剪模式

9 种裁剪模式非常容易理解，我们举例看看其中的几种。同样建议开发者参考上一小节中，我们想象的一个容器，这个容器用来裁剪图片的不同部位。

将 post-image 的 mode 属性设置为 top，效果如图 4-13 所示。

top 模式只保留图片的上部，裁剪掉了剩余部分。注意，这种模式不会缩放图片。我们可以仔细地再观察一下图中的图片，不仅仅是裁剪掉了图片的下部，上部水平方向也发生了裁剪。因为图片不会缩放，我们所设置的容器不能够在水平方向上完全把图片装进去，所以水平方向也发生了裁剪。这点是大家要注意的。

不同于 4 种缩放模式，裁剪模式是不会缩放图片的。用一张小图片来替换上面的大图，比如用 avatar 头像图片，替换后的效果如图 4-14 所示。

明显可以看到，由于图片的原始尺寸小于容器的尺寸，裁剪模式也不会使图片发生缩放，所以结果就是不会裁剪图片。

接着我们再将 mode 设置为 bottom right，效果如图 4-15 所示。

图片只被保留了右下角部分，其余部分全部被裁剪掉了。

其他几种裁剪类型从字面意思上都非常好理解，就不在这里一一列举了，开发者可以自行替换 mode 的属性值，看看裁剪效果。



图 4-13 mode = top



图 4-14 一张小图没有发生任何裁剪行为



图 4-15 mode=bottom right 时的效果

4.6 完成静态文章列表

先把上节更改的 `post-image` 的 `mode` 属性恢复成我们需要的 `mode = aspectFill`。

现在，文章列表还只有 1 篇文章。1 篇文章如何叫做文章列表？

为了多几篇文章，我们将代码清单 4-6 的代码再复制几份，依次加入到 `post.wxml` 文件中。这里再复制两份，形成一个有 3 篇文章的文章列表。

如果 CSS 代码编写足够健壮，无须更改 CSS 代码，重复复制 `post.wxml` 中的文章代码即可迅速新增文章，且样式不会错乱。保存后，模拟器将呈现出 3 篇一模一样的文章来。效果如图 4-16 所示。



图 4-16 复制文章

开发者可任意复制若干数量的文章，让页面看起来更像是一个文章列表。为了避免重复的数据，我们修改其中的两个文章数据，更改后的 post.wxml 文件如下。

代码清单 4-10 完整的文章列表页面代码

post.wxml

```
<view>
  <swiper vertical="{{false}}" indicator-dots="true" autoplay="true" interval="5000" circular="true">
    <swiper-item>
      <image src="/images/post/post-1@text.jpg" />
    </swiper-item>
    <swiper-item>
      <image src="/images/post/post-2@text.jpg" />
    </swiper-item>
    <swiper-item>
      <image src="/images/post/post-3@text.jpg" />
    </swiper-item>
  </swiper>
  <view class="post-container">
    <view class="post-author-date">
      <image src="/images/avatar/avatar-5.png" />
      <text>Jan 28 2017</text>
    </view>
```




```

<text class="post-title">小时候的冰棍儿与雪糕</text>
<image class="post-image" src="/images/post/post-4.jpg" mode="aspectFill" />
<text class="post-content">冰棍与雪糕绝对不是同一个东西。3 到 5 毛钱的雪糕犹如现在的哈根达
斯，而 5 分 1 毛的冰棍儿就像现在的老冰棒。时过境迁，...
</text>
<view class="post-like">
  <image src="/images/icon/wx_app_collect.png" />
  <text>108</text>
  <image src="/images/icon/wx_app_view.png"></image>
  <text>92</text>
  <image src="/images/icon/wx_app_message.png"></image>
  <text>7</text>
</view>
</view>
<view class="post-container">
  <view class="post-author-date">
    <image src="/images/avatar/avatar-1.png" />
    <text>Jan 9 2017</text>
  </view>
  <text class="post-title">从童年呼啸而过的火车</text>
  <image class="post-image" src="/images/post/post-5.jpg" mode="aspectFill" />
  <text class="post-content">小时候，家的后面有一条铁路。听说从南方北上的火车都必须经过这条
铁路。火车大多在晚上经过，但也不定是只有在夜深人静的时候，火车的声音才能从远方传来...
</text>
  <view class="post-like">
    <image src="/images/icon/wx_app_collect.png" />
    <text>108</text>
    <image src="/images/icon/wx_app_view.png"></image>
    <text>92</text>
    <image src="/images/icon/wx_app_message.png"></image>
    <text>7</text>
  </view>
</view>
<view class="post-container">
  <view class="post-author-date">
    <image src="/images/avatar/avatar-3.png" />
    <text>Jan 29 2017</text>
  </view>
  <text class="post-title">记忆里的春节</text>
  <image class="post-image" src="/images/post/post-1.jpg" mode="aspectFill" />
  <text class="post-content">年少时，有几样东西，是春节里必不可少的：烟花、新衣、凉菜、压岁
钱、饺子。年分大小年，有的地方是腊月二十三过小年，有的地方是腊月二十四.....
</text>

```

```

<view class="post-like">
  <image src="/images/icon/wx_app_collect.png" />
  <text>108</text>
  <image src="/images/icon/wx_app_view.png"></image>
  <text>92</text>
  <image src="/images/icon/wx_app_message.png"></image>
  <text>7</text>
</view>
</view>
</view>

```

保存后可以看到，3 篇不同的文章已出现在了页面中。开发者可自行调整代码中的文字、图片，无须和示例代码保持一致。

4.7 .js 文件的代码结构与 Page 页面的生命周期

到目前为止，我们还没有在页面的 js 文件中写过一行代码，是时候来学习一下小程序逻辑层代码的编写了。

如果开发者是使用 3.2 小节中介绍的快速新建页面文件的方法来创建的 post 页面，那么由开发工具生成的 post.js 文件内默认将包含代码清单 4-11 所示的代码。

代码清单 4-11 js 文件默认的代码结构

post.js

```

// pages/post/post.js
Page({
  data: {},
  onLoad: function(options){
    // 页面初始化 options 为页面跳转所带来的参数
  },
  onReady: function(){
    // 页面渲染完成
  },
  onShow: function(){
    // 页面显示
  },
  onHide: function(){
    // 页面隐藏
  },
  onUnload: function(){
    // 页面关闭
  }
})

```



页面 js 文件默认代码包含了我们可能使用到的代码结构, 整个页面执行了一个 `Page({...})` 方法, 参数是一个 `Object` 对象, 用来指定页面的初始数据 (`data`)、生命周期函数 (`on` 开头的函数)、事件处理函数等。

本节主要介绍页面的生命周期, 关于 `data` 变量和其他的事件处理函数, 后续章节再做详细介绍。

什么是页面的生命周期? 如同人的成长需要分为出生、童年、青年、中年、老年一样, 一个页面从创建到卸载, 同样会经历以下 5 个周期:

- 加载
- 显示
- 渲染
- 隐藏
- 卸载

MINA 框架分别提供了 5 个生命周期函数来监听这 5 个特定的生命周期, 以方便开发者可以在这些特定的时刻执行一些自己的代码逻辑, 它们分别是:

- `onLoad` 监听页面加载, 一个页面只会调用一次。
- `onShow` 监听页面显示, 每次打开页面都会调用。
- `onReady` 监听页面初次渲染完成, 一个页面只会调用一次, 代表页面已经准备妥当, 可以和视图层进行交互。
- `onHide` 监听页面隐藏。
- `onUnload` 监听页面卸载。

我们可以做一个小测试, 来了解生命周期函数的触发时机。向 `post.js` 的 5 个生命周期函数中添加以下代码:

代码清单 4-12 5 个生命周期函数

`post.js`

```
Page({
  data: {},
  onLoad: function(options){
    console.log("onLoad:页面被加载");
  },
  onShow: function(){
    console.log("onShow:页面被显示")
  },
  onReady: function(){
    console.log("onReady:页面被渲染")
  },
  onHide: function(){
    console.log("onHide:页面被隐藏")
  },
})
```



```

onUnload:function(){
  console.log("onUnload:页面被卸载")
}
})

```

保存代码，并将开发工具切换到【调试】→【Console】面板，编译看看 Console 的输出，如图 4-17 所示。

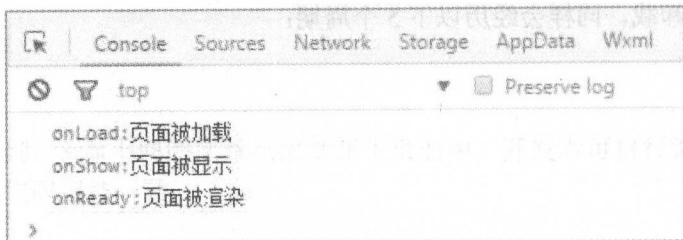


图 4-17 生命周期函数的执行顺序

可以看到，一个页面要正常显示，必须经历以上 3 个生命周期：加载、显示、渲染。注意这 3 个生命周期函数的执行顺序，首先是 onLoad，其次是 onShow，最后才是 onReady。

这里要特别提醒各位开发者，onShow 的执行时刻是在 onReady 之前的，但官方文档在编写时，将 onReady 放置在 onShow 之前（截止 0.11.122100 版本）。虽然只是顺序不同，但很容易让开发者误以为 onReady 是在 onShow 之前的，但这是不正确的。

那么 onHide 和 onUnload 呢？这两个函数的触发需要执行一些 API 的操作，比如当页面执行 navigateTo 方法或者使用小程序 tab 栏切换页面时会执行 onHide 函数；而当页面执行 redirectTo 或 navigateBack 的时候会执行 onUnload 函数。

除了以上 5 个生命周期函数外，还有以下 3 个小程序特定事件的处理函数：

- onPullDownRefresh 监听用户下拉动作的事件处理函数。
- onReachBottom 页面上拉触底事件的处理函数。
- onShareAppMessage 用户点击右上角分享。

开发者还可以添加任意的函数或数据到这个 Page 方法的 Object 参数中，在页面的函数中用 this 即可访问这些自定义函数或者数据。

关于 onHide 和 onUnload 以及 3 个特定事件的处理函数，我们将在后面介绍到导航、tab 栏、刷新、分享等项目需求时，再具体演示和讲解。放在具体的示例里演示，效果远比用文字理论描述要好。

官方文档中，还给出了一个较为全面的 Page 实例生命周期图解，如图 4-18 所示。

我们大概可以看到整个图分为左右两侧，左侧是视图层，右侧是服务逻辑层。整个页面的生命周期都是围绕着这两个层来进行的。他们之间不是孤立的，而是有很多的事件与通知交互的。目前，我们所学的知识还不足以完全解释页面的整个生命周期。

我们所讲的 5 个生命周期函数就在图 4-18 右侧多次出现，如果仔细观察，会发现以下几个特点：

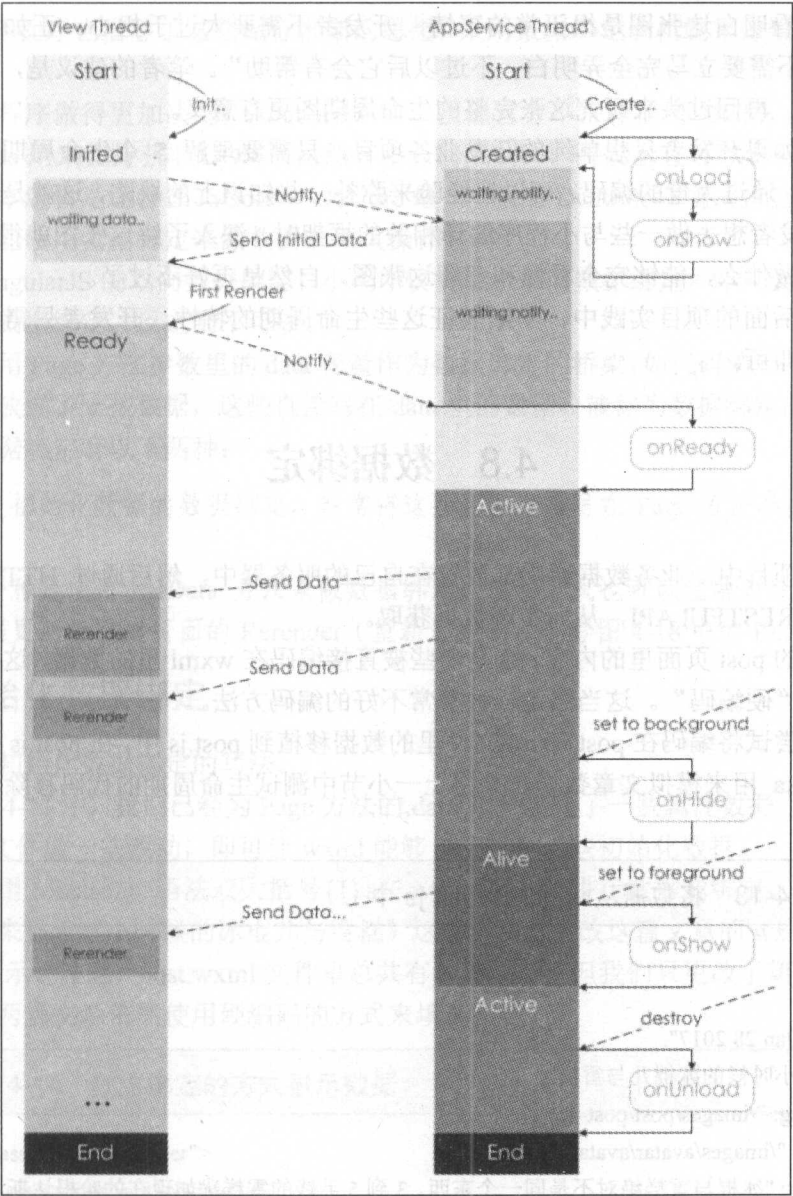


图 4-18 页面生命周期图解

- onLoad、onShow 和 onReady 确实是按照前面所讲到执行顺序依次执行。
- onLoad 与 onReady 在整个页面的生命周期中只会执行 1 次，除非这个页面被执行了 onUnload 卸载掉了（卸载掉后这个页面的生命周期就结束了）。
- onHide 与 onShow 在一次生命周期内可能会执行多次。
- 除了 First Render 第一次渲染，页面还有可能会 Rerender 再次渲染多次。数据更新会造成页面的重新渲染。开发者还要注意，小程序仅在第一次 First Render 完成后，提供了监听函数 onReady，对于以后的 Rerender 并没有提供相应的监听函数。所以，onReady 仅用来监听“第一渲染”完成。

现在无法看明白这张图是很正常的事情，开发者不需要太过于担心。正如官方文档中所说：“此图你不需要立马完全弄明白，不过以后它会有帮助”。笔者的建议是，当遇到问题或者业务需要时，再回过头来研究这张完整的生命周期图更有意义。

事实上，如果开发者只想单纯的开发业务项目，只需要理解 5 个生命周期函数发生的时机与意义即可。通过大量的编码，可以让经验来弥补一些知识上的缺陷，这就是所谓的熟能生巧。但如果开发者想去做一些与小程序编译相关的框架时，深入了解这张图就很有必要了。当然，无论你想做什么，能够完全看懂和理解这张图，自然是再好不过了。

我们会在后面的项目实践中，不断验证这些生命周期的特性，开发者只需要记得到时候回过头来看看即可。

4.8 数据绑定

在真实的项目中，业务数据通常都放置在自己的服务器中，然后通过 HTTP 请求来访问服务器提供的 RESTFUL API，从而实现数据获取。

现在我们的 post 页面里的内容，全是一些被直接编码在 wxml 里的数据，这样的代码写法我们通常称为“硬编码”。这当然是一种非常不好的编码方法。

我们现在尝试将编码在 post.wxml 文件里的数据移植到 post.js 中，在 post.js 中加入一个临时变量 postData 用来模拟文章数据，并将上一小节中测试生命周期的代码移除。编写完成后的代码如下。

代码清单 4-13 将数据从 wxml 移植到 js 中

post.js

```
Page({
  data: {
    date: "Jan 28 2017",
    title: "小时候的冰棍儿与雪糕",
    postImg: "/images/post/post-4.jpg",
    avatar: "/images/avatar/avatar-5.png",
    content: "冰棍与雪糕绝对不是同一个东西。3 到 5 毛钱的雪糕犹如现在的哈根达斯，而 5 分 1 毛的冰棍儿就像现在的老冰棒。时过境迁，...",
    readingNum: 92,
    collectionNum: 108,
    commentNum: 7
  },
})
```

那么如何将 data 中的这些数据“填充”到页面中，并显示这些数据呢？

如果是开发传统的网页，肯定会使用以下思路：首先获取到 HTML 文档的 DOM，然后对 DOM 标签进行赋值，从而实现数据的显示。但在小程序中，是没有 DOM 结构的，所以这个思路行不通。在许多流行的 MVC 或者 MVVM 框架中，比如 AngularJS、Vue.js 中，都有数据



绑定的概念。小程序也借鉴了这些流行框架的思想，采用数据绑定的机制来做数据的初始化和更新。

只不过小程序做得更加决绝。AngularJS 中，虽然官方不推荐使用 DOM，但至少还有一个内置的 jQLite 用来支持获取 DOM（虽然有很多的限制），开发者也可以自行集成 jQuery。但小程序的脚本逻辑是运行在 JSCore 中，JSCore 是一个没有 DOM 的环境，它完全抛弃了 DOM 结构，我们只能使用数据绑定来做数据的相关操作。

不同于 AngularJS 的双向数据绑定，小程序仅实现了单向数据绑定，即只支持从逻辑层传递到渲染层的数据绑定，反之则不可以。

小程序使用 Page 方法参数里的 data 变量作为数据绑定的桥梁。如代码清单 4-13 所示，data 里已经被我们放置了一些数据，这些直接写在 data 里的数据，被称为数据绑定的初始化数据。

注意，数据绑定有以下两种：

- 一种是初始化数据的数据绑定，通常将这些数据直接写在 Page 方法参数的 data 对象下面。
- 另外一种是使用 setData 方法来做数据绑定，这种方式也可以理解为数据更新。这样的数据更新将引起页面的 Rerender（重新渲染），参考图 4-18 中的 Rerender。

4.8.1 初始化数据绑定

先来看看初始化数据绑定的写法。

代码清单 4-13 中，我们已经为 Page 方法的 data 对象填充了一些属性数据。现在，只需要对 post.wxml 文件做一些改动，即可让 wxml 能够“接收”这些初始化数据。

小程序使用 Mustache 语法双大括号 {{}} 在 wxml 组件里进行数据的绑定。我们试着用数据绑定的方式来显示《小时候的冰棍儿与雪糕》这篇文章，更改这篇文章的 wxml 代码，如代码清单 4-14 所示。注意，post.wxml 文件里总共有 3 篇文章，但我们只更改了第一篇文章的相关代码，其他两篇文章依然使用硬编码的方式来填充数据。

代码清单 4-14 数据绑定的方式显示数据

post.wxml

```
<view class="post-container">
  <view class="post-author-date">
    <image src="{{avatar}}" />
    <text>{{date}}</text>
  </view>
  <text class="post-title">{{title}}</text>
  <image class="post-image" src="{{postImg}}" mode="aspectFill" />
  <text class="post-content">{{content}}</text>
  <view class="post-like">
    <image src="/images/icon/wx_app_collect.png" />
    <text>{{collectionNum}}</text>
    <image src="/images/icon/wx_app_view.png"></image>
    <text>{{readingNum}}</text>
  </view>
</view>
```

```

<image src="/images/icon/wx_app_message.png"></image>
<text>{{commentNum}}</text>
</view>
</view>

```

保存后可以看到，页面并没有变化，第一篇文章的数据正常地显示了出来，这说明数据绑定成功了。

可以看到双大括号`{{}}`中，写入了一些变量名。细心的开发者应该发现`{{}}`里的变量名称同js文件里data对象的属性名称是相同的。可见，数据绑定非常简单，只要将data对象的属性名填入到双大括号`{{}}`中即可。MINA框架会自动在运行时用data数据替换这些`{{}}`。比如`{{date}}`，在运行后将被替换为“Jan 28 2017”，而`{{readingNum}}`将被替换为“92”。

我们用图4-18页面生命周期图解这张图，解释一下初始化数据绑定的过程。

当页面执行了onShow函数后，逻辑层会收到一个通知(Notify)；随后逻辑层会将data对象以json的形式发送到View视图层(Send Initial Data)，视图层接收初始化数据后，开始渲染并显示初始化数据(First Render)，最终将数据呈现在开发者的眼前。

这里需要注意，如果数据绑定是作用在组件的属性中，比如`<image src="{{avatar}}" />`，则一定要在`{{}}`外边加上双引号，否则小程序会报错。

如果是内容型的数据绑定，则不需要加双引号，比如`<text>{{date}}</text>`。

数据绑定的Mustache语法还有一些其他用法，我们依然会放在下面的实例项目中来讲解。

4.8.2 在哪里可以查看数据绑定对象

开发工具为我们提供了一个面板专门用来查看和调试数据绑定变量，这个面板就是在第2章中介绍的AppData面板。

我们来看一下Orange Can项目此时的AppData情况。打开【调试】→【AppData】，可以看到以下的数据情况，如图4-19所示。



图4-19 post页面在AppData面板中的数据绑定情况

请开发者注意，AppData面板对于调试和理解数据绑定有非常重要的作用，建议当开发者遇到数据绑定相关问题时，一定要首先打开这个面板来查看具体的数据绑定情况。



AppData 下的数据以页面为组织单位。因为现在只在 post 页面里做了数据绑定，所以 AppData 下边只出现了 pages/post/post 这一个页面的数据。如果同时有多个页面进行了数据绑定，那么这里将出现多个页面的数据绑定情况。

如图 4-19 所示，可以看到在 pages/post/post 下显示了 post 页面的数据绑定变量情况，它的属性和 post.js 文件中所设置的 data 对象属性是一模一样的。

可以在这里更改某一项数据的值。更改是实时进行的，改变任何一个值，开发工具都能实时地将变化更新到模拟器 UI 里显示。开发者可以自行尝试，更改一下 title、date 或者 content 等的属性值，并注意观察模拟器的 UI 变化。

这里还有一个小技巧，让页面的数据以 json 的形式呈现：点击图 4-19 中的【Tree】这个选项，将打开如图 4-20 所示的面板。点击【Code】后，数据将以 json 的形式呈现，如图 4-21 所示。

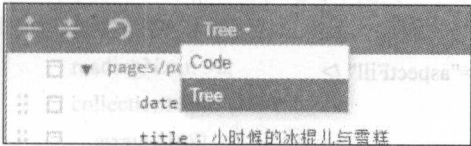


图 4-20 切换数据呈现形式

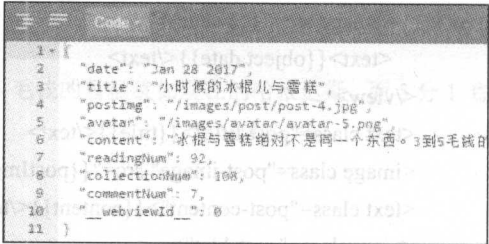


图 4-21 以 json 的格式呈现数据

json 格式的数据，非常利于我们快速复制这些数据。

4.8.3 绑定复杂对象

4.8.1 小节中的 Page 参数下的 data 对象只是一个最简单的 js 对象，它的属性值都只是简单的文本与数字。在实际项目中，可能出现较为复杂的对象，将 data 对象更改为如下代码：

代码清单 4-15 较为复杂的 data 对象

post.js

```
Page({
  data: {
    object: {
      date: "Jan 28 2017"
    },
    title: "小时候的冰棍儿与雪糕",
    postImg: "/images/post/post-4.jpg",
    avatar: "/images/avatar/avatar-5.png",
    content: "冰棍与雪糕绝对不是同一个东西。3 到 5 毛钱的雪糕犹如现在的哈根达斯，而 5 分 1 毛的冰棍儿就像现在的老冰棒。时过境迁...",
    readingNum: 92,
    collectionNum: {
      array: [108]
    }
  },
})
```

```
commentNum: 7
},
})
```

此时，data 对象已不再是简单的对象，它的属性还包含有对象和数组。运行代码后，我们发现，小程序并不会报错，但 UI 上的数据无法正确显示。原因是被绑定的 data 对象数据结构改变后，相应的也需要在页面的 wxml 里做出和 data 数据结构等同的调整。调整之后的代码如下（注意，我们同样只更改第一篇文章的相关代码）：

代码清单 4-16 根据 data 对象的结构调整 wxml

post.wxml

```
<view class="post-container">
  <view class="post-author-date">
    <image src="{{avatar}}" />
    <text>{{object.date}}</text>
  </view>
  <text class="post-title">{{title}}</text>
  <image class="post-image" src="{{postImg}}" mode="aspectFill" />
  <text class="post-content">{{content}}</text>
  <view class="post-like">
    <image src="/images/icon/wx_app_collect.png" />
    <text>{{collectionNum.array[0]}}</text>
    <image src="/images/icon/wx_app_view.png"></image>
    <text>{{readingNum}}</text>
    <image src="/images/icon/wx_app_message.png"></image>
    <text>{{commentNum}}</text>
  </view>
</view>
```

现在，date 数据的绑定语法由 {{date}} 变成了 {{object.date}}；而 collection 数据的绑定语法由 {{collectionNum}} 变成了 {{collectionNum.array[0]}}。这些相应的调整都是根据 data 数据结构的变化做出的，开发者请仔细对比。

重新运行项目，文章数据又可以正常显示了。

4.8.4 数据绑定更新

还可以使用 setData 函数来做数据绑定，这种方法可以理解为“数据更新”。setData 方法位于 Page 对象的原型链上：Page.prototype.setData。大多数情况下，我们使用 this.setData 的方式来调用这个方法。

setData 的参数接受一个对象，以 key 和 value 的形式将 this.data 中的 key 对应的值设置成 value。

上面这句话要注意两点：

- setData 会改变 this.data 变量里相同 key 的值。



- setData 执行后会通知逻辑层执行 Rerender，并立刻重新渲染视图，参考图 4-18。

说起来好像很难理解，但使用起来非常简单，来看看具体代码。

在 post 页面中新增一个 onLoad 函数，并在其中执行 setData，更改后的代码如下：

代码清单 4-17 使用 setData 更新数据

post.js

```
Page({
  data: {
    object: {
      date: "Jan 28 2017"
    },
    title: "小时候的冰棍儿与雪糕",
    postImg: "/images/post/post-4.jpg",
    avatar: "/images/avatar/avatar-5.png",
    content: "冰棍与雪糕绝对不是同一个东西。3 到 5 毛钱的雪糕犹如现在的哈根达斯，而 5 分 1 毛的冰棍儿就像现在的老冰棒。时过境迁，...",
    readingNum: 92,
    collectionNum: {
      array: [108]
    },
    commentNum: 7
  },
  onLoad: function() {
    this.setData({
      title: "一根雪糕的经济学原理"
    })
  }
})
```

运行后我们发现，第一篇文章的标题由 data 里所设置的 title: "小时候的冰棍儿与雪糕"，被更改成了“一根雪糕的经济学原理”，但其他的数据并没有改变。原因在于我们使用 this.setData 只更新了 title 这一个数据，并未改变其他诸如 date、avatar、content 等数据。

此外，当执行了代码清单 4-17 后，此时 this.data.title 的值将是“一根雪糕的经济学原理”，而不再是“小时候的冰棍儿与雪糕”，因为 this.setData 的执行也会改变 this.data 里的值。

这就是 setData 的基本用法。此外，setData 参数中的 key 是非常灵活的，来看看 key 可能出现的形式。修改代码清单 4-17 中的 onLoad 方法如下：

代码清单 4-18 字符串作为 key

post.js

```
onLoad: function () {
  this.setData({
    "title": "一根雪糕的经济学原理"
  })
}
```

key 可以使用字符串来表示，代码如下：

代码清单 4-19 key 的形式举例 1

post.js

```
onLoad: function () {
  this.setData({
    "collectionNum.array[0]": 66
  })
}
```

更改 collectionNum 数组子元素的值，代码如下：

代码清单 4-20 key 的形式举例 2

post.js

```
onLoad: function () {
  this.setData({
    "object.date": "Jan 28 2019"
  })
}
```

更改 object 下的 date 的数值。

用 this.setData 所绑定或者更新的数据，并不要求在 this.data 中已预先定义。看看下面的例子，将 post.js 文件中的代码改为代码清单 4-21 所示。

代码清单 4-21 使用 setData 直接设置数据绑定

post.js

```
Page({
  data: {
  },
  onLoad: function () {
    var iceCreamData = {
      object: {
        date: "Jan 28 2017"
      },
      title: "小时候的冰棍儿与雪糕",
      postImg: "/images/post/post-4.jpg",
      avatar: "/images/avatar/avatar-5.png",
      content: "冰棍与雪糕绝对不是同一个东西。3 到 5 毛钱的雪糕犹如现在的哈根达斯，而 5 分 1 毛的冰棍儿就像现在的老冰棒。时过境迁，...",
      readingNum: 92,
      collectionNum: {
        array: [108]
      },
    },
```




```
commentNum: 7
}
this.setData({
  postData: iceCreamData
})
}
```

上述代码中，去掉了 `this.data` 中的初始化数据，转而直接使用 `this.setData` 进行数据更新，从而实现数据绑定，这种方法也是可行的。

但这时项目并不能正常运行，UI 上第一篇文章变成了一篇空白，且没有任何错误提示。原因在于，数据绑定的数据结构变了，`wxml` 里的 `{{}}` 也需要做相应的改变。

借助我们之前讲到的【AppData】面板来看一下现在的数据绑定情况，如图 4-22 所示。

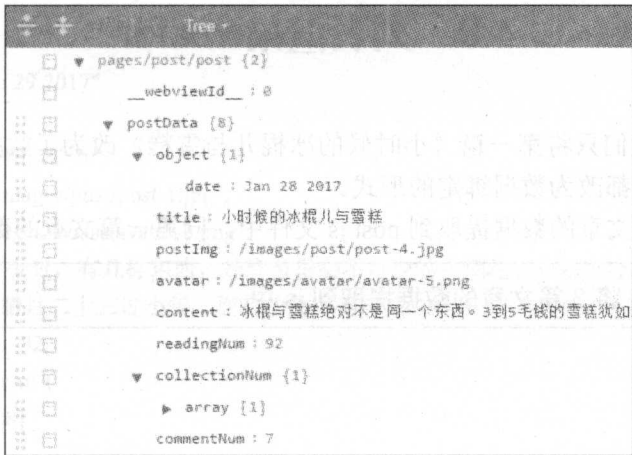


图 4-22 AppData 里的数据

很明显，这个数据结构和之前的是不一样的。所有的属性都被 `postData` 对象包裹了起来，因为我们在 `this.setData` 的时候指定的 `key` 是 `postData`，`value` 是文章的数据。所以，`wxml` 里的 `{{}}` 需要如下这么改：

代码清单 4-22 更改 `wxml{{}}` 的数据绑定

post.wxml

```
<view class="post-container">
  <view class="post-author-date">
    <image src="{{postData.avatar}}"/>
    <text>{{postData.object.date}}</text>
  </view>
  <text class="post-title">{{postData.title}}</text>
  <image class="post-image" src="{{postData.postImg}}" mode="aspectFill"/>
  <text class="post-content">{{postData.content}}</text>
  <view class="post-like">
    <image src="/images/icon/wx_app_collect.png"/>
```

```

<text>{{postData.collectionNum.array[0]}}</text>
<image src="/images/icon/wx_app_view.png"></image>
<text>{{postData.readingNum}}</text>
<image src="/images/icon/wx_app_message.png"></image>
<text>{{postData.commentNum}}</text>
</view>
</view>

```

只需要在每个`{{}}`里加入 `postData` 即可。比如`{{title}}`应当改为`{{postData.title}}`。

请各位开发者注意，关于数据绑定的错误，小程序目前不会给出任何的错误提醒。如果你发现整个页面是空白的又没有错误消息，多半是数据绑定出了问题。这个时候 `AppData` 面板是最好的调试工具。

4.9 列表渲染 wx:for

到目前为止，我们只将第一篇《小时候的冰棍儿与雪糕》改为了数据绑定的形式，现在来尝试把所有的文章都改为数据绑定的形式。

首先将其他两篇文章的数据提取到 `post.js` 文件中，同第一篇文章的数据组成一个数组。

代码清单 4-23 将 3 篇文章的数据提取到 js 中

post.js

```

Page({
  data: {
  },
  onLoad: function () {
    var postList = [{
      object: {
        date: "Jan 28 2017"
      },
      title: "小时候的冰棍儿与雪糕",
      postImg: "/images/post/post-4.jpg",
      avatar: "/images/avatar/avatar-5.png",
      content: "冰棍与雪糕绝对不是同一个东西。3 到 5 毛钱的雪糕犹如现在的哈根达斯，而 5 分 1 毛
的 冰棍儿就像现在的老冰棒。时过境迁，...",
      readingNum: 92,
      collectionNum: {
        array: [108]
      },
      commentNum: 7
    },
    {
      object: {

```



```

    date: "Jan 9 2017"
  },
  title: "从童年呼啸而过的火车",
  postImg: "/images/post/post-5.jpg",
  avatar: "/images/avatar/avatar-1.png",
  content: "小时候，家的后面有一条铁路。听说从南方北上的火车都必须经过这条铁路。火车大
多在晚上经过，但也不定是只有在夜深人静的时候，火车的声音才能从远方传来...",
  readingNum: 92,
  collectionNum: {
    array: [108]
  },
  commentNum: 7
},
{
  object: {
    date: "Jan 29 2017"
  },
  title: "记忆里的春节",
  postImg: "/images/post/post-1.jpg",
  avatar: "/images/avatar/avatar-3.png",
  content: "年少时，有几样东西，是春节里必不可少的：烟花、新衣、凉菜、压岁钱、饺子。年
分大小年，有的地方是腊月二十三过小年，而有的地方是腊月二十四...",
  readingNum: 92,
  collectionNum: {
    array: [108]
  },
  commentNum: 7
},
]
this.setData({
  postList: postList
})
}
})

```

注意，代码中 `this.setData` 的 `key` 更改为了 `postList`，`value` 被更换成了一个包含 3 个元素的数组，每个元素代表一篇文章的数据。

现在，我们已经有 3 篇文章的数据了。我们当然可以像改写第一篇文章一样，依次改写其他两篇文章的 `{{}}` 绑定。但这样好吗？

这里来考虑一个问题，如果我有 100 篇文章，怎么办？难道也像这样手动地去填写 100 篇文章的 `{{}}` 吗？

如果可以在 `wxml` 里做 `for` 循环该有多好？

小程序确实提供了一个 `wxml` 组件的 `for` 循环，称为列表渲染。我们一起来看看，如何使

用列表渲染来改写文章列表，先给出改写后的代码。

代码清单 4-24 使用列表渲染改写文章列表

post.wxml

```
<view>
  <swiper vertical="{{false}}" indicator-dots="true" autoplay="true" interval="5000" circular="true">
    <swiper-item>
      <image src="/images/post/post-1@text.jpg" />
    </swiper-item>
    <swiper-item>
      <image src="/images/post/post-2@text.jpg" />
    </swiper-item>
    <swiper-item>
      <image src="/images/post/post-3@text.jpg" />
    </swiper-item>
  </swiper>
  <block wx:for="{{postList}}" wx:for-item="item" wx:for-index="idx">
    <view class="post-container">
      <view class="post-author-date">
        <image src="{{item.avatar}}" />
        <text>{{item.object.date}}</text>
      </view>
      <text class="post-title">{{item.title}}</text>
      <image class="post-image" src="{{item.postImg}}" mode="aspectFill" />
      <text class="post-content">{{item.content}}</text>
      <view class="post-like">
        <image src="/images/icon/wx_app_collect.png" />
        <text>{{item.collectionNum.array[0]}}</text>
        <image src="/images/icon/wx_app_view.png"></image>
        <text>{{item.readingNum}}</text>
        <image src="/images/icon/wx_app_message.png"></image>
        <text>{{item.commentNum}}</text>
      </view>
    </view>
  </block>
</view>
```

重点关注<block></block>这对括号内的代码。<block>标签没有实质意义，它并不是组件，所以我们称作“标签”，它仅仅是一个包装，不会在页面内被渲染，可以理解为常见编程语言里的括号，在 block 标签中被包裹的元素将被重复渲染。

在 block 标签上，放置了一个 wx:for 的特殊属性，它的值为{{postList}}。wx:for 将绑定一个数组，在本示例中，这个数组就是 postList，它对应 post.js 文件中 setData 的数组数据。

wx:for-item 指定数组当前元素的变量名，我们将当前元素的变量名指定为 item。



`wx:for-index` 指定当前元素在数组中序号的变量名，我们命名为 `idx`。当然，在本示例中，只是定义了这个 `wx:for-index`，并没有真正地使用它。

有了 `item` 这个数组子元素后，就可以按照上一小节中改写第一篇文章时所使用的 `{{}}` 语法来填写数据绑定了。在所有的 `{{}}` 填写数据绑定变量。保存运行后，发现三篇文章都可以正常显示，但代码的总量却大大减少了。

`wx:for` 并不是一定要作用在 `block` 标签上，如果把代码清单 4-24 中的 `block` 标签换成 `view`，一样可以正常运行。但并不推荐使用 `view` 等组件来做列表渲染。因为同 HTML 一样，我们希望标签或者组件元素是语义明确的。`view` 组件通常被用来当作容器或者是区域分隔，它有它的使命，不应该被滥用。

开发者可以尝试将 `wx:for-item="item"` 属性给去掉，文章列表依然可以正常显示。不定义 `item`，那么 `{{}}` 内的 `item` 是哪里来的？事实上，不定义数组子元素的变量名，小程序默认子元素的变量名就是 `item`。如果你不喜欢 `item` 这个变量名，可以将它替换为其他你喜欢的变量名，比如 `wx:for-item="element"`。如果更改了子元素的变量名，记得将 `{{}}` 中对应的 `item` 都更换为被指定的变量名，比如 `element`。

4.10 配置单个页面导航栏背景色

注意观察 `post` 文章页面，它顶部的导航栏并不是默认的黑色，而是呈现出和 `welcome` 欢迎页面相同的橘红色。

原因在于我们在 `app.json` 中配置了全局导航栏的颜色为 `#b3d4d`。在项目设计图里，全局导航栏的配色应该是 `#4A6141`，所以，我们现在将全局导航栏的颜色配置为 `#4A6141`。在 `app.json` 中更改全局导航栏配色，代码如下：

代码清单 4-25 更改全局导航栏配色

app.json

```
"window": {
  "navigationBarBackgroundColor": "#4A6141"
}
```

更改后发现，`welcome` 页面顶部的导航栏颜色也被更换成了新的颜色。所以，我们需要单独配置 `welcome` 页面的导航栏颜色，让它不受全局配置的影响。

全局配置是在 `app.json` 中设置，那么对单个页面的配置应该在页面的 `json` 文件中配置。在 `welcome.json` 中添加如下代码：

代码清单 4-26 配置页面的导航栏颜色

welcome.json

```
{
  "navigationBarBackgroundColor": "#ECC0A8"
}
```

保存后发现 `welcome` 页面的导航栏颜色已经被更改成了橘红色。

那么页面的 json 文件配置和 app.json 文件的配置有什么不同？

- 页面的 json 文件只能够配置和 window 相关的属性。window 属性的配置项请参考 3.9 小节。但 app.json 除了可以配置 window 外还可以配置 pages、tabBar 等选项。
- 页面的 json 配置不需要像 app.json 那样，加上 window 这个对象，直接编写 window 下面的配置项即可。请仔细对比代码清单 4-25 和代码清单 4-26 的区别。

修改完成后，以后再新建任何页面，页面的导航栏背景色都将被配置为 #4A6141 这个颜色。当然，可以将颜色修改为任何你喜欢的颜色。

4.11 从欢迎页面跳转到文章页面

我们现在一共编写了两个页面：welcome 欢迎页面与 post 文章页面。来尝试将两个页面连接起来，通过点击 welcome 页面的“开启小程序之旅”跳转到 post 文章页面。

首先将 welcome 页面重新调整为启动页面，代码如下：

代码清单 4-27 将 welcome 页面设置为启动页

post.js

```
{
  "pages": [
    "pages/welcome/welcome",
    "pages/post/post"
  ],
  "window": {
    "navigationBarBackgroundColor": "#ECC0A8"
  }
}
```

调整启动页面的方法很简单，将启动页面的路径放在 pages 数组下的第一个元素即可。

4.11.1 事件

要从 welcome 页面跳转到 post 页面，需要使用事件来响应点击“开启小程序之旅”这个动作。

什么是事件？

严肃一些的定义是：事件是视图层（wxml）到逻辑层（js）的通信方式。简单一些理解，事件可以让我们在 js 里处理一些用户在界面上的一些操作并对这些操作做出反馈。比如点击 welcome 页面“开启小程序之旅”按钮后，需要在 js 里调用 MINA 框架的 API，使页面从 welcome 跳转到 post。

要实现这样的机制，需要做两件事情：

- 在组件上注册事件。注册事件将告诉小程序，我们要监听哪个组件的什么事件。在本例中，需要监听“开启小程序之旅”这个组件的 `tap` 事件。
- 在 `js` 中编写事件处理函数响应事件。也就是说，监听到事件后，需要编写自己的业务。在本例中，我们将调用 MINA 框架的导航 API，让 `welcome` 页面跳转到 `post` 页面。

更改 `welcome.wxml` 页面的代码，如代码清单 4-28 所示。

代码清单 4-28 添加 `tap` 事件`welcome.wxml`

```
<view class="container">
  <image class="avatar" src="/images/avatar/avatar-1.png"></image>
  <text class="motto">Hello, 桔子罐头</text>
  <view catchtap="onTapJump" class="journey-container">
    <text class="journey">开启小程序之旅</text>
  </view>
</view>
```

和之前的代码相比并没有太大的改动，仅仅是在 `class="journey-container"` 的这个 `view` 组件上添加了一个 `catchtap="onTapJump"` 的事件绑定。事件绑定的写法同组件的写法相同。它的意思是，监听点击这个动作，当用户点击这个动作后，将执行一个 `onTapJump` 的函数，这个函数必须在页面的 `js` 中定义。下面的代码定义了 `tap` 事件的处理函数。

代码清单 4-29 添加 `tap` 操作的事件处理函数`welcome.js`

```
Page({
  onTapJump: function (event) {
    wx.redirectTo({
      url: "../post/post",
      success: function () {
        console.log("jump success")
      },
      fail: function () {
        console.log("jump failed")
      },
      complete: function () {
        console.log("jump complete")
      }
    });
  }
})
```

代码中为 `Page` 方法的 `Object` 参数定义了一个函数：`onTapJump`。函数的名称可以任意指定，但必须和代码清单 4-28 中定义的 `catchtap="onTapJump"` 保持一致。当用户点击或者触碰“开启小程序之旅”这个按钮后，MINA 框架将执行 `onTapJump` 这个函数，并将一个 `event`

对象作为参数传递到函数里。

保存运行代码，点击“开启小程序之旅”，页面将从 welcome 欢迎页面跳转到 post 文章页面。

4.11.2 redirectTo 与 navigateTo

在上一小节中，我们在 onTapJump 函数里调用了 wx.redirectTo 方法从而实现了页面跳转。小程序共提供了 3 个导航 API，以帮助开发者实现页面跳转。

- wx.redirectTo
- wx.navigateTo
- wx.switchTab(122100 版本新增)

他们之间的区别是：redirectTo 将关闭当前页面，跳转到指定页面；navigateTo 将保留当前页面，跳转到指定页面；而 switchTab 只能用于跳转到带 tabBar 的页面，并关闭其他所有非 tabBar 页面。

switchTab 页面将在后面学习 tabBar 选项卡时再具体介绍，本节主要来看看 redirectTo 和 navigateTo 的区别。

redirectTo 和 navigateTo 在使用方式上完全相同，他们都接受一个 Object 对象作为参数。Object 对象中最重要的属性是 url，它将指定要跳转的页面路径。

请注意 url 是页面的路径，不要加上文件的扩展名（如同 app.json 中定义 pages 一样）。如果在页面路径后加上一个“.wxml”，比如将 url 设置为 url: “../post/post.wxml”，页面无法跳转，并会报错。

Object 参数还可以接收 3 个方法，分别是：

- success 跳转页面成功时 MINA 框架将调用此函数。
- fail 跳转页面失败时 MINA 框架将调用此函数。
- complete 无论成功或者失败，MINA 框架都将调用此函数。

具体写法可参考代码清单 4-29。

将这 3 个方法拿出来单独列举是因为在小程序中，几乎所有异步类型的 API 都配备有这 3 个方法。比如后面要学习的操作反馈 API: wx.showToast，获取用户信息 API: wx.getUserInfo 等。在以后的其他 API 学习过程中我们就不再一一列举这 3 个方法了。

再次保存并运行以上代码，点击跳转后，页面将跳转到 post 文章页面。此时我们发现没有办法再返回到 welcome 页面了。这就是 redirectTo 的特点，它将卸载 welcome 页面，并执行页面的 onUnload 事件函数。可以来验证一下，在 welcome.js 里加入一个 onUnload 函数和一个 onHide 函数。

代码清单 4-30 加入 onUnload 和 onHide 函数

welcome.js

```
onUnload: function (event) {
  console.log("page is unload")
},
```




```
onHide: function (event) {
  console.log("page is hide")
},
```

运行代码，发现 Console 将输出“page is unload”，但并不会输出“page is hide”。

再看看 navigateTo。将代码清单 4-29 中的 wx.redirectTo 更改为 wx.navigateTo。保存运行代码后将发现，navigateTo 跳转到 post 页面后，页面左上角有一个返回按钮，如图 4-23 所示。点击返回后还可以再返回到 welcome 页面。除此之外，navigateTo 将执行 onHide 事件回调，并输出了“page is hide”。

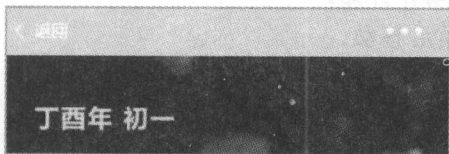


图 4-23 左上角有【返回】按钮

所以，redirectTo 将关闭当前页面并将页面卸载；而 navigateTo 仅仅会隐藏当前页面，还可以再次返回到被隐藏的页面。这是他们最重要的区别。

再来考虑一个问题。当 navigateTo 跳转到 post 页面后，再次从 post 页面返回到 welcome 页面时，post 页面会执行 onHide 还是 onUnload 呢？答案是会执行 post 页面的 onUnload 函数（不能保证以后的版本是否还会更改，但目前的 130400 版本确实是会执行 onUnload 函数）。也就是说，当从子页面返回到父页面时，子页面会被卸载。开发者可以仿照 welcome 页面自行验证。

但事实上，子页面执行 onUnload 函数的行为是从 122100 版本后才更改的，在之前的版本中子页面返回到父页面并不会执行 onUnload 函数，造成大量的子页面残留在小程序中。这在当时给开发者带来了巨大的困扰，还好官方在后续版本中更改了这个行为。页面是否被卸载是非常重要的行为，不卸载页面将使全局性的一些行为，比如音乐播放的处理，变得非常复杂。所以，了解这些页面的生命周期对于开发者来说是很重要的，否则极易引起 bug。

我们还可以再试试 wx.switchTab 这个方法，将 welcome.js 中的 wx.navigateTo 更改为 wx.switchTab，其他保持不变，运行一下代码。

页面无法执行跳转，且 Console 将输出 jump failed。原因之前我们解释过，switchTab 只能跳转到带有 tabBar 选项卡的页面，而 post 页面并不带有选项卡，所以无法执行跳转。tabbar 的配置将在后面讲到，现在开发者无须关心。

现在，我们暂时选取 navigateTo 作为跳转方法。

4.11.3 小程序最多只能有 5 层页面

当我们使用 navigateTo 从父页面跳转到子页面后，就形成了 2 个页面层级。可以继续在于子页面里使用 navigateTo 跳转到子页面。

但小程序里强制规定，只允许有最多五层父子页面。所以请开发者注意，尽量避免多层次的交互方式。事实上，太多的子页面将严重影响用户的产品体验。建议页面最多不要超过 3 层。

redirectTo 不存在这个问题，因为当跳转到另一个页面后，上一个页面被强制卸载掉了。

4.11.4 冒泡事件与非冒泡事件

什么是冒泡事件？

冒泡事件指某个组件上的时间被触发后，事件还会向父级元素传递；父级元素还会继续向父级的父级传递，一直到页面的顶级元素。而非冒泡事件则不会向父级元素传递事件。

在 4.11.1 小节中，我们使用了 tap 事件，监听点击或者触摸动作，而 tap 是一个冒泡事件。常见的冒泡事件类型还有下面几种：

- touchstart 手指触摸动作开始。
- touchmove 手指触摸后移动。
- touchcancel 手指触摸动作被打断，如来电提醒、弹窗。
- touchend 手指触摸动作结束。
- tap 手指触摸后马上离开。
- longtap 手指触摸后，超过 350ms 再离开。

相对于 PC 上的 Web 浏览器，小程序的事件并不多。需要注意的是，在 wxml 组件里注册事件时，不可以直接使用 tap="function" 或 touchmove="function"，需要在事件名之前添加 catch 或者 bind 前缀。比如在 welcome 页面跳转时，我们就使用了 catchtap 而并没有直接使用 tap。

bind 和 catch 有什么区别？

区别在于，对于以上几个冒泡事件，catch 将阻止事件继续向父节点传播，而 bind 不会阻止事件的传播。

基本上所有的组件都有以上这些冒泡事件。

除以上 6 种事件外，如无特殊申明都是非冒泡事件，非冒泡事件大多不是通用事件，而是某些组件特有的事件。如 <form/> 的 submit 事件，<input/> 的 input 事件，<scroll-view/> 的 scroll 事件等。



第 5 章

模块、模板与缓存

本章是小程序的进阶内容。主要介绍了模块、模板和缓存的概念以及使用方法。模板是小程序中的重点和难点，它将大幅度地提高代码的复用性与可维护性，避免开发者编写重复的代码。

本章也特别指出了模板与组件的区别，小程序仅仅实现了模板化而不能自定义组件，这是非常遗憾的一件事儿。

缓存的应用也是小程序中的一个特色，开发者的很多业务都需要借助缓存来实现，比如用户的令牌、城市列表数据等都可以写入小程序的缓存中。

本章我们还分别尝试使用 ES5 和 ES6 语法编写“数据库”访问类，开发者可以自行体会一下 ES6 编写 Class 的优越性。

5.1 将文章数据从业务中分离

现在，所有的文章数据都被强行写在 `post.js` 里，这污染了我们的业务层。我们尝试将这些数据分离到一个单独的 `js` 文件中。

在 `Orange Can` 项目的根目录下新建一个文件夹，命名为 `data`。然后在 `data` 目录下新建一个 `js` 文件，命名为 `data.js`。

将 `post.js` 文件中 `onLoad` 函数下的 `postList` 数组数据整体剪切到 `data.js` 文件中，并将其中的 `collectionNum` 和 `date` 等数据改为最简单的字符串（此前为了演示复杂对象的数据绑定，我们在【4.8.3 绑定复杂对象】这一小节中将 `collectionNum` 和 `date` 改为了对象的形式）。新的 `data.js` 文件代码如下：

代码清单 5-1 将 `postList` 数据剪切到 `data.js` 文件中

`post.js`

```
var postList = [{
  date: "Jan 28 2017",
  title: "小时候的冰棍儿与雪糕",
  postImg: "/images/post/post-4.jpg",
  avatar: "/images/avatar/avatar-5.png",
  content: "冰棍与雪糕绝对不是同一个东西。3 到 5 毛钱的雪糕犹如现在的哈根达斯，而 5 分 1 毛的冰棍儿就像现在的老冰棒。时过境迁，...",
  readingNum: 0,
  collectionNum: 0,
  commentNum: 0
},
{
  date: "Jan 9 2017",
  title: "从童年呼啸而过的火车",
  postImg: "/images/post/post-5.jpg",
  avatar: "/images/avatar/avatar-1.png",
  content: "小时候，家的后面有一条铁路。听说从南方北上的火车都必须经过这条铁路。火车大多在晚上经过，但也不定是只有在夜深人静的时候，火车的声音才能从远方传来...",
  readingNum: 0,
  collectionNum: 0,
  commentNum: 0
},
{
  date: "Jan 29 2017",
  title: "记忆里的春节",
  postImg: "/images/post/post-1.jpg",
  avatar: "/images/avatar/avatar-3.png",
```



```

content: "年少时，有几样东西，是春节里必不可少的：烟花、新衣、凉菜、压岁钱、饺子。年分
大小年，有的地方是腊月二十三过小年，而有的地方是腊月二十四..."，
readingNum: 0,
collectionNum: 0,
commentNum: 0

```

5.2 小程序的模块

上一小节中我们提取的数据文件 `data.js` 可以视作是小程序的一个模块，但现在还没有办法从其他文件访问这个模块。

我们还需要使用 `module.exports` 向外部暴露一个接口。在 `data.js` 文件的最下部添加以下代码：

代码清单 5-2 向外部暴露模块接口

data.js

```

module.exports = {
  postList: postList
}

```

定义好模块后，接下来就可以在其他 `js` 文件中引用这个模块。我们需要在 `post.js` 中引入 `data.js` 这个模块。

代码清单 5-3 引入模块

post.js

```

var dataObj = require("../data/data.js");

Page({
  data: {
  },
  onLoad: function () {
    this.setData({
      postList: dataObj.postList
    })
  }
})

```

代码第一行的 `require(path)` 将模块引入到 `post.js` 中，并将模块对象赋值给 `dataObj`。随后在 `onLoad` 函数里取出 `postList` 数据，并进行数据绑定。

使用 `require` 引用 `js` 模块时，要特别注意以下几点：

- 被引用的文件一定要带有扩展名 `js`，这一点是不同于页面路径的。



- path 路径不可以使用绝对路径，否则会报错。应该使用相对路径。
- 在 JavaScript 文件中声明的变量和函数只在该文件中有效，不同的文件中可以声明相同名字的变量和函数，不会互相影响。

所以，如果使用 `require('/data/data.js')`，小程序会找不到 `data.js` 这个文件。

注意为什么是 `dataObj.postList`? 因为在输出模块时，我们是将 `postList` 作为一个 `object` 的属性赋值给 `module.exports` 的，参考代码清单 5-2。所以在 `require` 时，得到的也是一个 `object` 并非是 `postList`，需要使用 `dataObj.postList` 才能获取到真实的文章数据。

这样的做的好处是，`object` 不仅可以包含 `postList`，你还可以在 `data.js` 文件中定义除 `postList` 外的其他数据，并作为 `object` 的属性一起输出。

我们在上一小节中更改了 `postList` 的数据结构，所以要调整 `post.wxml` 里 `{{}}` 的语法才可以正常显示数据。

代码清单 5-4 调整 wxml 中 `{{}}` 数据绑定语法

post.wxml

```
<block wx:for="{{postList}}" wx:for-item="item" wx:for-index="idx">
  <view class="post-container">
    <view class="post-author-date">
      <image src="{{item.avatar}}" />
      <text>{{item.date}}</text>
    </view>
    <text class="post-title">{{item.title}}</text>
    <image class="post-image" src="{{item.postImg}}" mode="aspectFill" />
    <text class="post-content">{{item.content}}</text>
    <view class="post-like">
      <image src="/images/icon/wx_app_collect.png" />
      <text>{{item.collectionNum}}</text>
      <image src="/images/icon/wx_app_view.png"></image>
      <text>{{item.readingNum}}</text>
      <image src="/images/icon/wx_app_message.png"></image>
      <text>{{item.commentNum}}</text>
    </view>
  </view>
</block>
```

保存运行代码，项目正常地显示出了 3 篇文章数据。

事实上，`require` 只是模块化的一种方式。还可以使用 ES6 的 `Module` 来编写模块。开发工具默认使用 `babel` 将开发者的 ES6 代码转化成 ES5 代码。



5.3 小程序的模板化

使用列表渲染展现文章列表是最好的方法吗？恐怕不是。如果其他页面同样需要显示文章列表怎么办？把代码清单 5-4 中的代码到处拷贝吗？这当然是最差的选择。

借助一下函数这个思想。我们通常会将一些公共的、经常使用的业务逻辑提取成一个公共的函数，当在多个地方需要使用函数时，只需要调用这个函数即可完成相应的业务。使用函数的好处是不言而喻的。

事实上，有一句话是这么描述软件开发的：编程世界里遇到的绝大多数问题都可以用封装的思想来解决。夸张一点儿来说，你所能看到的代码，其实全是封装过的代码。

小程序也提供了一个称作模板的技术来支持对 wxml 组件的封装，但是这种封装仅仅只是 wxml 的代码片段，并没有实现像 AngularJS 里的完整模块儿。在 AngularJS 里 HTML、js 可以作为一个整体被封装起来。但是在小程序中，我们只能将 wxml 封装，无法将模板的业务逻辑（js）也封装起来。

首先来看看如何使用模板，随后再讨论封装模板业务逻辑的问题。

要使用模板，自然需要先新建模板文件。在/pages/post下新建目录 post-item，作为模板文件目录。接着在该目录下新建 2 个文件：post-item-tpl.wxml 和 post-item-tpl.wxss。这里使用 tpl 来结尾，只是一种建议和习惯，并不是强制要求，开发者可以自行定义模板名称。

使用模板是为了简化 post.wxml 中文章的写法，让文章可以成为一个单独的“组件”（但不是真的组件，只是模板），供其他多个地方使用。想想我们在使用 image、text 等组件时是不是很简单，只需要一个简单的<image></image>就可以实现图片的显示功能。同样，我们也可以尝试将文章编写成一个“组件”。

现在，尝试将 post.wxml 中<block>标签中关于文章的代码剪切到 post-item-tpl.wxml 中，让这段代码成为一个可复用的“组件”。

代码清单 5-5 编写文章模板

post-item-tpl.wxml

```
<template name="postItemTpl">
  <view class="post-container">
    <view class="post-author-date">
      <image src="{{item.avatar}}"/>
      <text>{{item.date}}</text>
    </view>
    <text class="post-title">{{item.title}}</text>
    <image class="post-image" src="{{item.postImg}}" mode="aspectFill"/>
    <text class="post-content">{{item.content}}</text>
    <view class="post-like">
      <image src="/images/icon/wx_app_collect.png"/>
      <text>{{item.collectionNum}}</text>
      <image src="/images/icon/wx_app_view.png"></image>
    </view>
  </view>
</template>
```

```

<text>{{item.readingNum}}</text>
<image src="/images/icon/wx_app_message.png"></image>
<text>{{item.commentNum}}</text>
</view>
</view>

```

</template>模板相关内容必须被包裹在<template></template>标签内,使用 name 属性指定 template 模板的模板名。这个模板名将在引用模板时被使用。

当定义好一个 template 后,可以在其他页面引用这个 template。现在我们在 post.wxml 中引用并使用这个 template。

代码清单 5-6 引用并使用 postItemTpl

post.wxml

```

<import src="post-item/post-item-tpl.wxml"/>
//.....省略若干其他代码
<block wx:for="{{postList}}" wx:for-item="item" wx:for-index="idx">
  <template is="postItemTpl" data="{{item}}" />
</block>

```

注意,以上代码没有列出 swiper 组件的代码。限于篇幅,只列出和当前内容相关的代码,请开发者保留 post 页面中的 swiper 代码。包括在以后的内容中,我们也只会列出页面的部分代码,这点请开发者注意。

保存运行,可以看到文章列表可以正常地加载和显示。我们来分析一下上面这段代码是如何引用和使用 template 模板的。

在 post.wxml 的顶部使用<import src="templatePath" />来引用模板。对于 templatePath 路径,这里需要注意,在当前版本中,可以在后面加 wxml 文件扩展名,也可以不加扩展名。但官方示例中是带有.wxml 扩展名的,所以建议开发者带上模板文件的扩展名。

引用后模板就可以在页面中使用这个模板了。在需要模板的位置使用 template 标签引入模板。template 的 is 属性指定要使用哪个模板,这里我们当然要使用 postItemTpl 这个模板。

再次类比一下函数,函数通常可以定义若干个参数,并从函数调用方传入一些数据。同样,模板也可以传入数据。通过 template 的 data 属性,可以向 template 传递数据。这里将 wx:for 得到的 item 传入到 template 里,这样就可以在 template 内部使用这个 item 了。要注意的是,向模板里传入数据,同样要使用{{}}的数据绑定语法,比如 data={{item}}。

5.4 消除 template 模板对外部变量名的依赖

来看一个有趣儿的问题。我们之前讲过,列表渲染中 wx:for-item 可以指定数组子元素的变量名。现在,尝试将代码清单 5-6 中的 wx:for-item="item"改成 wx:for-item="item1"。这将使 postList 数组子元素的变量名由 item 变成 item1。

此时,如果要将数据传入到 postItemTpl 中,则应该设置 data="item1"。



做完以上变更后，再次保存运行代码，会发现文章数据将消失，并且没有任何错误提示。没有显示数据，肯定是有问题的，再次强调很多时候数据绑定的相关问题，小程序不会做任何的错误提示。

那么，问题出在什么地方？之前代码可以正常运行是因为我们向 `template` 传入的变量名 `data="{{item}}"`，恰好和 `template` 里面数据绑定的变量名 `item` 一样，开发者可以回顾一下代码清单 5-5 和代码清单 5-6。但一旦更改了 `item` 为 `item1` 后，`template` 就找不到这个 `item` 了。

类比一下函数，函数的参数名是可以由函数自己自定义的，这保证了函数不受外部变量名的影响。但是 `template` 模板却并没有提供一个定义参数名的地方，没有办法更改从外部传入的 `item1` 为 `item`。

当然，可以通过将 `postItemTpl` 这个 `template` 内部的 `item` 更改为 `item1` 来让代码重新正常运行，开发者可以自行尝试一下。

但这并不是一个好的办法，看起它非常的蠢。我们讲过模板的好处是它可以让多个调用方来调用，不可能要求每个调用方都使用同样的变量名来调用模板，这种由定义方要求调用方遵守变量名命名的做法是不太合理的。

要解决这个问题，就必须消除 `template` 对于外部变量名的依赖，可以使用扩展运算符“...”展开传入对象变量来消除这个问题。

将 `post.wxml` 中使用模板的地方更改为：

代码清单 5-7 使用扩展运算符展开对象

```
<template is="postItemTpl" data="{{...item}}" />
```

接着去掉 `post-item-tpl.wxml` 文件中 `{{}}` 里所有的 `item`。

代码清单 5-8 去掉 `{{}}` 中所有的 `item`

post-item-tpl.wxml

```
<template name="postItemTpl">
  <view class="post-container">
    <view class="post-author-date">
      <image src="{{avatar}}" />
      <text>{{date}}</text>
    </view>
    <text class="post-title">{{title}}</text>
    <image class="post-image" src="{{postImg}}" mode="aspectFill" />
    <text class="post-content">{{content}}</text>
    <view class="post-like">
      <image src="/images/icon/wx_app_collect.png" />
      <text>{{collectionNum}}</text>
      <image src="/images/icon/wx_app_view.png"></image>
      <text>{{readingNum}}</text>
      <image src="/images/icon/wx_app_message.png"></image>
      <text>{{commentNum}}</text>
    </view>
  </view>
</template>
```

```

</view>
</view>
</template>

```

保存并运行，文章列表可以正常显示了。

{{...item}} 可以将 item 这个对象展开。展开之后再传入到 template 里，就可以保证 template 不再依赖 item 这个变量名。

5.5 include 与 import 引用模板的区别

5.3 小节中，介绍了如何使用 import 来引用模板。小程序还提供了另外一种引入模板的方式：include。

include 在使用上同 import 有以下区别：

- import 需要先引入 template，然后再使用 template；但 include 不需要预先引入，直接在需要的地方引入模板即可。
- include 模式非常简单，就是简单的代码替换，不存在作用域，也不能像 import 一样使用 data 传递变量。

如果要在 post.wxml 中使用 include，我们需要做一些改动。

代码清单 5-9 使用 include 引入模板

post.wxml

```

<block wx:for="{{postList}}" wx:for-item="item" wx:for-index="idx">
  <include src="post-item/post-item-tpl.wxml"/>
</block>

```

以上代码将 block 标签中的 template 更换成了<include>。include 同样使用 src 属性指向模板文件。

更改以上代码后，界面无法显示任何数据。来排查下问题，首先判断模板文件有没有正常地加载到页面中，可以使用调试中的【wxml】面板来看一下 post.wxml 中有没有 postItemTpl 的相关代码，如图 5-1 所示。

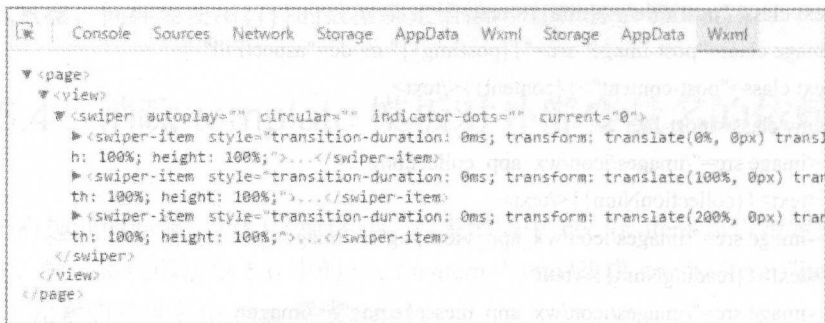


图 5-1 wxml 中没有显示 postItemTpl 的代码



并没有显示 `postItemTpl` 的相关代码。

为什么呢？

原因在于 `include` 无法引入包含有 `template` 标签的代码，而现在 `post-item-tpl.wxml` 里的所有代码都是被 `template` 标签包裹起来的。如果想使用 `include` 代替 `import`，那么模板文件内就不能使用 `template` 标签。

将 `post-item-tpl.wxml` 里的 `template` 标签删除，只保留文章本身的 `wxml` 代码。再次使用调试下的【`wxml`】查看 `post.wxml` 页面，发现 `post-item-tpl.wxml` 代码已被成功地引入到了 `post.wxml` 中。

但现在文章的数据并没有显示出来，原因在于 `include` 无法向模板里传递变量，它仅仅只是一个占位符，当小程序运行时，会用 `include` 的 `src` 属性所指向的文件替换 `include` 自身。这一点同样可以从调试下的【`wxml`】面板里看到。

回顾一下代码清单 5-9，假想 `include` 已经被替换成了 `post-item-tpl.wxml` 里的代码，那么要想显示出文章数据，就必须再一次在 `post-item-tpl.wxml` 中的 `{{}}` 中加入 `"item"` 这个变量名，就像代码清单 5-5 所做的那样，开发者可自行尝试一下。

除了在使用上有所不同，`include` 和 `import` 还存在其他不同之处：`import` 有作用域的概念，即只会 `import` 目标文件中定义的 `template`，而不会 `import` 目标文件 `import` 的 `template`。

例如：`C import B`，`B import A`，在 `C` 中可以使用 `B` 定义的 `template`，在 `B` 中可以使用 `A` 定义的 `template`，但是 `C` 不能使用 `A` 定义的 `template`。

而 `include` 就很简单了，它只是一个占位符，仅做简单的代码替换。

那么，何时使用 `include`？何时使用 `import`？

这里笔者的建议是，如果模板仅仅是静态 `wxml`，不涉及数据的传递，可以使用 `include`。但如果模板涉及数据绑定，还是建议使用 `import`。所以，这里选择使用 `import` 来引用 `postItemTpl` 模板。

5.6 CSS 的模块化

在之前的几个小节里，已经成功地将 `wxml` 代码做成了模板。既然是模板，就应该有模板的样式。我们当然可以维持现在的代码不做改变，因为现在整个项目运行正常，但这样并不合理。样式同样应该作为模板的一部分被“打包”起来。

将 `post.wxss` 中同文章相关的样式（所有以 `post-` 开头的样式）全部剪切到 `post-item-tpl.wxss` 文件中，`post.wxss` 文件只留下 `swiper` 组件相关的样式。

代码清单 5-10 `postItemTpl` 的样式

`post-item-tpl.wxss`

```
.post-container {
  /*... 省略部分代码*/
}
```

```
/*... 省略部分代码*/
```

```
.post-like text {  
  /*... 省略部分代码*/  
}
```

保存运行，发现 post 页面的样式乱掉了。

在定义了 postItemTpl 后，我们需要在 post.wxml 中引用它。同样，当定义了模板的 wxss 文件后，也需要在 post.wxss 文件中引用它。

引用样式文件的语法是 @import "src"。在 post.wxss 文件的顶部添加如下代码：

代码清单 5-11 引入模板样式文件

post.wxss

```
@import "post-item/post-item-tpl.wxss";
```

在引入 CSS 文件时，既可以是以上代码中所使用的相对路径，也可以是绝对路径。保存后，文章列表的样式恢复正常。

5.7 令人遗憾的模板化而非组件化

在组件化的编程思维里，一个前端组件必须要同时满足视图层代码的组件化和逻辑层代码的组件化。在小程序里勉强可以实现 view 的组件化（template 模板），但是模板却不可以实现业务逻辑 js 代码的组件化。

开发者可以自行尝试一下，如果在/pages/post/post-item 下再新建一个 post-item-tpl.js 文件，尝试将此文件作为模板的业务逻辑代码是行不通的，小程序无法自动运行这个 js 文件。

也就是说，小程序只实现了模板化而并没有实现组件化。当然，我们同样可以像引入 wxss 样式文件一样，在 post.js 中通过 require 来引用模板的 js 文件，这样可以将模板的业务代码集中在一个模块儿中。

但对比一下官方提供的 image、swiper 等组件，他们的业务逻辑并不需要在 js 代码里引入，而是被很好地封装起来了，我们只需要在 wxml 中添加这些组件就可以很好地应用组件。同时，这些组件的数据交互也是通过组件的自定义属性来传递的，比如 swiper 组件的 autoplay 和 interval 等属性。

很期待官方能够支持自定义组件，而不仅仅是现在的模板。自定义组件将大幅提高代码的复用性，并且使业务代码变得更加简洁。在官方的一个 Q&A 列表里，提到了打算支持自定义组件，但并没有说明准确的时间点。



5.8 使用缓存在本地模拟服务器数据库

在之前的小节中，我们将文章相关数据分离到了 `data.js` 文件中，并在 `post.js` 文件里通过 `require` 来加载 `data.js` 文件。

引用并读取 `data.js` 当然没有问题，但我们考虑一个问题，如果要修改数据怎么办？修改后的数据，还想共享给其他页面使用，并长期保存这些数据怎么办？

比如，在后面的内容中，我们会增加文章的评论、阅读量计数、文章收藏数等动态计数功能；且当用户重启应用后这些用户数据并不应该丢失。

我们需要一个类似于数据库的概念，可以读取、保存、更新这些数据，且这些数据不会因为应用程序重启或者关闭而消失。

小程序提供了一个非常重要的特性——缓存，来支持这样的特性。

现在，将 `data.js` 这个文件视作是本地数据库的初始化数据，要做的第一件事就是将这些初始化数据“装进”缓存中，以形成数据库的初始化数据。

5.8.1 应用程序的生命周期

在什么时候将初始化数据装载到缓存中是一个需要考虑的问题。考虑一下，初始化的行为在整个应用程序生命周期里只应该发生一次，所以最好的时机是在小程序启动时来装载初始化数据。

应用程序启动时是一个 MINA 框架行为，如果想掌握这个时刻，并做一些我们想做的事儿，就需要框架通知我们：嗨，现在是应用程序启动的时候，你要做什么事儿，就在这个函数里做吧。

想想之前页面的生命周期，每一个重要的结点，MINA 框架都会给页面一个通知，比如 `onLoad`、`onShow`、`onReady` 等。同样，整个应用程序也有自己的生命周期。

还是类比一下页面的生命周期。在页面的 JS 文件中，我们使用 `Page(object)` 来注册页面，并在 `object` 中指定页面的生命周期函数等。同样，可以在 `app.js` 文件中使用 `App(object)` 来注册小程序，并在 `object` 中指定小程序的生命周期函数等。

`Object` 参数有以下几个：

- `onLaunch` 监听小程序初始化，当小程序初始化完成时，会触发 `onLaunch`（全局只触发一次）。
- `onShow` 监听小程序显示，当小程序启动，或从后台进入前台显示，会触发 `onShow`。
- `onHide` 监听小程序隐藏，当小程序从前台进入后台，会触发 `onHide`。
- `onError` 错误监听函数，当小程序发生脚本错误，或者 API 调用失败时，会触发 `onError` 并带上错误信息。

当然，除了以上几个 MINA 框架给予的特定函数，开发者还可以添加任意函数或数据到 `Object` 参数中，用 `"this"` 可以访问这些函数和数据。



这里特别对 onShow 和 onHide 做一个说明。onHide 会在小程序从前台进入后台时触发，比如在 iPhone 中通过按下“Home”键，将微信隐藏时触发 onHide；而 onShow 不仅仅在小程序启动时会触发，还会在小程序从后台到前台时触发，相当于是 onHide 的反向动作。

可以在开发工具中模拟应用程序的“进入后台”和“从后台显示”这两个动作，从而触发 onShow 和 onHide。开发工具提供了一个【后台】按钮，点击后应用程序将模拟进入后台的效果，再点击一次将从后台返回到前台，如图 5-2 所示。



图 5-2 开发工具模拟小程序进入后台和返回前台的操作

5.8.2 使用 Storage 缓存初始化本地数据库

上一小节中我们分析了，最好的初始化数据库的时机是在应用程序启动时，在 app.js 中加入以下代码：

代码清单 5-12 设置数据缓存

app.js

```
var dataObj = require("data/data.js")
```

```
App({
  onLaunch: function () {
    wx.setStorage({
      key: 'postList',
      data: dataObj.postList,
      success: function(res){
        // success
      },
      fail: function() {
        // fail
      },
      complete: function() {
        // complete
      }
    })
  },
})
```



在上面的代码中，首先通过 `require` 加载 `data.js` 文件作为初始化数据。在应用程序生命周期函数 `onLaunch` 里，使用 `wx.setStorage` 方法将初始化数据存入到小程序的缓存中。

什么是缓存？

缓存让小程序具备了本地存储数据的能力，它具有以下几个特点：

- 只要用户不主动清除缓存，则缓存一直存在。
- 缓存以 `key:value` 键值对的形式存在，很类似于服务器流行的 `memcache` 或者 `redis` 缓存型数据库。
- 小程序提供了一系列 API 用来操作缓存，包括：存储、读取、移除、清除全部和获取缓存信息。每种操作同时都具有同步和异步两个方法。具体 API 请参考官方文档。
- 请注意移除和清除的区别。删除某一个 `key` 的缓存，请使用 `wx.removeStorage` 方法；而如果想清除所有的缓存请使用 `wx.clearStorage` 方法。
- 要注意，小程序的缓存永久存在，不存在过期时间这个概念。如果想清除缓存，则需要主动调用清除缓存的 API。
- 小程序的本地缓存有容量上限，最大不允许超过 10MB。

代码清单 5-12 中的 `wx.setStorage(object)` 是一个异步方法，参数 `object` 包含 `key`，`data` 和 `success`、`fail`、`complete` 这 3 个通用方法（关于这 3 个通用方法，之前我们反复强调，几乎所有小程序的异步 API 方法中都包含这 3 个方法，后面的内容将不再列出这些方法，请开发者根据自己的需求来使用这些方法）。

`key` 用来设置缓存的键，而 `data` 用来设置缓存的值，可以是 JavaScript 对象或者字符串。

运行以上代码并不会出现明显的效果，但我们可以调试下的【Storage】面板里看看有什么变化，如图 5-3 和图 5-4 所示。

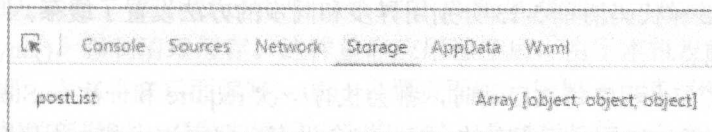


图 5-3 Storage 面板数据情况（1）

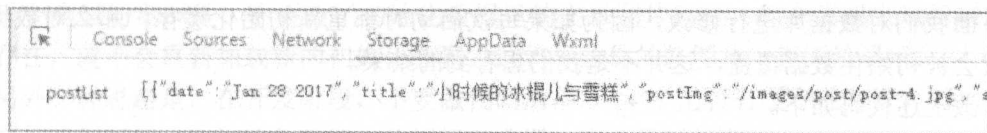


图 5-4 Storage 面板数据情况（2）

图 5-3 中的 `postList` 就是在代码中设置的 `key:'postList'`，后面的 `Array` 数组就是设置的 `data` 对象，也就是要初始化的数据，对应的是 `data.js` 文件的 3 篇文章数据。点击展开 `Array`，如图 5-4 印证了这一点。【Storage】面板是查看缓存的重要功能，当你遇到与缓存相关的问题时，请一定要到这里来看一看。

是的，这就是我们搭建的一个简易本地数据库，它具有增、删、改的功能。当然，它也具备简单的查询功能，但并不如 `MySQL` 这类数据库的查询功能强大。注意，将本地缓存理解为一个简易数据库的思想非常重要，我们应当像在服务器编写数据库访问类一样，编写一组操

作自己业务缓存的通用方法，而且最好将这些方法集中在一个“类”中。这样的做法将大大提高代码的可阅读性与可维护性。在实际项目中，本地缓存是非常重要的功能，可以极大地改善用户体验。

所有的缓存操作方法还有一个同步的版本，用同步的方法来改写一下代码清单 5-12。

代码清单 5-13 同步设置缓存

app.js

```
var dataObj = require("data/data.js")

App({
  onLaunch: function () {
    wx.setStorageSync('postList', dataObj.postList);
  },
})
```

同步方法 `wx.setStorageSync` 是在异步方法名 `wx.setStorage` 后加了一个后缀“Sync”。不仅仅是 `setStorage`，小程序中几乎所有同步方法的方法名都是在异步方法名后增加了“Sync”。

同步方法的参数非常简单，它接收 2 个参数，例如

```
wx.setStorageSync(key, data)
```

同步方法没有 `success`、`fail`、`complete` 等回调方法。在本书的后续代码中如果没有特殊情况，通常都用同步方法。开发者可以根据自己的业务和环境选取异步方法。但要注意的是，选取异步方法会大大增加代码风险率和调试难度。如果没有必要（通常是处于性能和体验的考虑），建议优先考虑同步方法。

代码清单 5-12 和代码清单 5-13 分别用异步和同步的方法设置了缓存。但考虑一下上面的代码还有没有问题。

上面的代码将在小程序每次启动时，都会执行一次 `require` 和一次 `setStorage`。但实际上，缓存如果不主动清除，它是一直存在的，因此完全没有必要每次启动小程序时都执行一次初始化数据库。仅当缓存不存在时，执行一次上述代码即可。

下面我们对数据库进行修改，因为如果每次启动时都重新初始化缓存，那么对数据库的修改就会被初始化数据覆盖，这并不是我们想看到的结果。

修改上述代码如下。

代码清单 5-14 优化缓存初始化判断

app.js

```
App({
  onLaunch: function () {
    var storageData = wx.getStorageSync('postList');
    if(!storageData){
      //如果 postList 缓存不存在
      var dataObj = require("data/data.js")
      wx.clearStorageSync();
    }
  }
})
```




```
wx.setStorageSync('postList', dataObj.postList);
```

```
}  
},  
})
```

`wx.getStorageSync(key)`这个方法可以获取指定 `key` 的缓存内容。如果指定 `key` 的缓存不存在，则说明数据库还没有初始化。那么此时首先使用 `wx.clearStorageSync()`清除所有的缓存数据，接着再重新读取并设置初始化数据。

以上代码优化了初始化缓存数据库的方案。只有当缓存数据库不存在时，才通过 `require` 加载 `data.js` 文件，并初始化数据库。这样可以避免每次启动应用程序都重复初始化数据库。

虽然通常来说，`require` 都是放在代码文件的顶部，但我们也可以在需要的时候才引用它。代码清单 5-14 中演示了这种用法。

本地缓存数据库，我们就初步建立完成了，后续内容我们还会持续完善这个数据库。

5.8.3 缓存的强制清理及注意事项

除了使用 `wx.clearStorageSync()`代码清除缓存外，在模拟器中还可以通过开发工具左侧的【缓存】工具进行缓存清理。【缓存】工具点击后会弹出 4 个选项，其中【清除数据存储】就是清除 `Storage` 的功能。

但要特别注意，真机上没有类似于开发工具这样的强制清理缓存的按钮。微信自带的缓存清理并不是用来清除小程序缓存的，这点要特别注意。

笔者在实际开发过程中遇到很多缓存引起的问题，其中大多数是因为更新了初始化数据后，却忘记在手机上清除缓存，以至于没办法更新真机上的初始化数据。

建议的解决方案是，在开发过程中，临时在页面里增加一个按钮，点击按钮执行 `wx.clearStorageSync()`，强制清理缓存。这样重启应用程序后，由于本机没有缓存，所以会重新加载初始化数据。本书将在后面编写 `setting` 设置页面时，增加一个清理缓存的选项。

在处理缓存相关问题时，开发者要保持头脑清醒，否则有时候一个小小的缓存没更新的问题，将浪费开发者大量的时间。

一个典型的案例是，你更改了初始化数据里的文章图片路径，但在真机上运行时，由于缓存存在，就不会重新加载新的初始化数据。这将导致你的新图片一直无法显示。

另外一种思路是，在开发阶段，不要做代码清单 5-14 中是否有缓存的判断，每次应用程序重启都强制更新一次初始化数据，从而保证数据一直是最新状态。

5.9 编写缓存数据库操作类

我们来构建一个访问缓存数据库的访问“类”。在 JavaScript 编程的世界里似乎“类”这个概念一直都不是那么流行，相当一部分原因在于 JavaScript 的面向对象和我们在大学和工作后所理解的诸如 Java、C#这种经典的面向对象语言有很多的不同，这是由于 JavaScript 历史原

因造成的。但 JavaScript 里并不是没有面向对象，只不过它是用原型链的方式来实现对象的继承机制。

ES6 的出现让 JavaScript 这个语言重新焕发了新生，module、lambda、class 等特性的支持，让 JavaScript 更加现代化。

考虑到本书主要内容是讲解小程序，如果全部使用 ES6，必然会全面使用面向对象的思想来构建整个项目，这会给部分不熟悉 ES6 的开发者造成一定的困扰。所以 Orange Can 项目的编写并没有全面使用 ES6，但 ES6 的重要性是不言而喻的，Orange Can 将尝试对于某些模块使用 ES6 来编写。

不使用 ES6 并不代表我们没有办法编写面向对象的代码，我们将尝试用 prototype 和 ES6 的 Class 分别来构建缓存数据库的操作类。

在项目根目录下新建 db 文件夹，并在该文件夹下新建 DBPost.js 文件，并在文件中写入以下代码：

代码清单 5-15 prototype 构建数据操作类

DBPost.js

```
var DBPost=function () {
    this.storageKeyName='postList';//所有的文章本地缓存存储键值
}

DBPost.prototype={

    //得到全部文章信息
    getAllPostData:function(){
        var res = wx.getStorageSync(this.storageKeyName);
        if(!res){
            res= require('../data/data.js').postList;
            this.execSetStorageSync(res);
        }
        return res;
    },

    //本地缓存 保存/更新
    execSetStorageSync:function(data){
        wx.setStorageSync(this.storageKeyName,data);
    },
};

module.exports = {
    DBPost:DBPost
};
```



上述代码首先定义了一个 DBPost 构造函数。在构造函数中，我们将 post 数据在缓存数据库中的 key, postList, 赋值给构造函数的 this 变量。注意，这个 postList 必须同 app.js 中我们初始化数据库时设置的文章数据的 key 相同，否则无法读取数据。

随后，我们在构造函数的原型链上添加一个对象，这个对象的所有属性和方法都会被构造函数的实例继承。比如，我们在这个对象中增加了一个 getAllPostData 方法，这个方法将可以获取缓存数据库中所有的文章数据。

在 getAllPostData 中，我们做了一个判断，如果缓存不存在将重新加载 data.js 数据文件，并存入到缓存数据库中。

最后，还是使用 module.exports 将 DBPost 输出。

如果开发者对面向对象的 JavaScript 不是太熟悉，建议去学习一下面向对象的 JavaScript 编程，网上有不少这方面的资料。

当然这里最好的写法还是使用 ES6 的 Class 和 Module 来编写。使用构造函数和 prototype 原型链构建对象，总是会让那些熟悉 Java、C# 等现代经典面向对象的开发者觉得很奇怪。ES6 的 Class 优化了 JavaScript 的对象构建方式，让对象看起来更加符合现代的面向对象写法。但是，ES6 大多特性只是一种语法糖，在本质上 JavaScript 的运行和解析机制并没有被改变。所以理解 JavaScript 的构造函数与原型链 prototype 依然非常重要。

5.10 使用缓存数据库操作类

现在，我们尝试在 post.js 中使用上一小节中定义的数据库操作类，将 post.js 代码更改一下。

代码清单 5-16 使用 DBPost

post.js

```
var DBPost = require('../db/DBPost.js').DBPost;
```

```
Page({
  data: {
  },
  onLoad: function () {
    var dbPost= new DBPost();
    this.setData({
      postList: dbPost.getAllPostData()
    });
  },
})
```

需特别注意的是，这里没有直接使用 require 加载 data.js 文件，因为 data.js 现在只是初始化数据，它已经在 app.js 中被装载到缓存数据库中。所以，我们现在 require 的是 DB 操作类所在的模块文件，通过这个类来操作文章数据。

代码第一行同样使用 require 加载 DBPost.js 文件，并读取 DBPost。



那么，如果要使用 DBPost，必须先使用操作符“new”将 DBPost 实例化。实例化 DBPost 后，就可以调用该对象的 getAllPostData 方法，从而读取所有文章的缓存数据并绑定到 postList 中。

保存后，程序可以正常地运行。

5.11 使用 ES6 改写缓存操作类

我们用 ES6 的新特性 Class、Module 来改写缓存操作类。

代码清单 5-17 用 ES6 改写 DBPost

DBPost.js

```
class DBPost {
  constructor(url) {
    this.storageKeyName='postList';
  }

  //得到全部文章信息
  getAllPostData(){
    var res = wx.getStorageSync(this.storageKeyName);
    if(!res){
      res= require('../data/data.js').postList;
      this.initPostList(res);
    }
    return res;
  }

  // 保存或者更新缓存数据
  execSetStorageSync(data){
    wx.setStorageSync(this.storageKeyName,data);
  }
};
export {DBPost}
```

以上代码使用 ES6 的 Class 改写了缓存数据库操作类。可以看到，同 prototype 的实现方式相比，ES6 的写法更加符合现代语言的类的标准定义习惯。

注意 Class 中定义的两个函数，它们是不需要 function 关键字的。同时，方法之间不要加“，”，否则会报错。

最后 export 输出语法也非常简洁，如 export {DBPost};

接着，我们再看如何使用 ES6 版本的 DBPost。



代码清单 5-18 使用 ES6 版本的 DBPost

post.js

```
import { DBPost } from '../db/DBPost.js';
```

```
Page({
  data: {
  },
  onLoad: function () {
    var dbPost = new DBPost();
    this.setData({
      postList: dbPost.getAllPostData()
    });
  },
})
```

注意，这里不再使用 `require` 来加载 `DBPost.js` 这个文件，而是使用 ES6 导入模块的关键字 `import` 将 `DBPost` 导入进来。这种写法的可读性比 `require` 要高得多。

通过以上两种不同的写法，可以清晰地看到 ES6 提供的 Class 模板让 JavaScript 的面向对象编程变得更加的清晰，更符合现代面向对象写法。但 Class 仅仅是一个语法糖，不使用 ES6 一样可以实现面向对象的编写方法。

建议开发者使用 Class 来编写对象，小程序本身在编译时会集成 `babel` 这个插件，将 ES6 转成 ES5。我们将选用 ES6 实现的这个版本作为 `Orange Can` 的项目代码，当然如果你不熟悉 ES6，也可以使用 `prototype` 这种编写方式。

5.12 完善文章数据

在这一章的末尾，我们将完善文章数据，顺便看一下如何更新缓存中的初始化数据。目前，我们仅有 3 篇文章的数据，现在再增加两篇文章的数据。

在 `data.js` 文件中再增加两篇文章数据，代码如下：

代码清单 5-19 增加两篇文章数据

data.js

```
{
  date: "Sep 22 2016",
  title: "换个角度，再来看看微信小程序的开发与发展",
  postImg: "/images/post/post-2.jpg",
  avatar: "/images/avatar/avatar-2.png",
  content: "前段时间看完了雨果奖中短篇获奖小说《北京折叠》。很有意思的是，张小龙最近也要把应用折叠到微信里，这些应用被他称为：小程序...",
  readingNum: 0,
```



```

collectionNum: 0,
commentNum: 0
},
{
  date: "Dec 28 2016",
  title: "2017 微信公开课 Pro",
  postImg: "/images/post/post-3.jpg",
  avatar: "/images/avatar/avatar-4.png",
  content: "在今天举行的 2017 微信公开课 PRO 版上，微信事业群总裁张小龙宣布，微信“小程序”
将于 1 月 9 日正式上线。",
  readingNum: 0,
  collectionNum: 0,
  commentNum: 0
},

```

理论上，当增加了两篇文章数据后，不需要增加任何代码，再次刷新项目后，文章列表页面应该立刻出现这两篇文章的数据。

但实际上并不是这样，保存后 post 页面还是只有 3 篇文章的数据。原因很简单，因为缓存已经存在了，如果更改了初始化数据又不清除缓存，那么缓存是不会被更新的。

点击开发工具左侧的【缓存】按钮，随后在弹出的菜单中点击【清除数据缓存】，然后再编译项目，发现文章列表里已经有 5 篇文章了。

5.13 完整的 data.js 数据

在这个章节的末尾，我们给出 data.js 文件的所有数据内容，包括已经用到的和没有用到的。

代码清单 5-20 完整的 data.js 文件

data.js

```

var postList = [{
  date: "Jan 28 2017",
  title: "小时候的冰棍儿与雪糕",
  postImg: "/images/post/post-4.jpg",
  avatar: "/images/avatar/avatar-5.png",
  content: "冰棍与雪糕绝对不是同一个东西。3 到 5 毛钱的雪糕犹如现在的哈根达斯，而 5 分 1 毛的冰棍儿就像现在的老冰棒。时过境迁...",
  readingNum: 23,
  collectionNum: 3,
  commentNum: 0,
  author: "林白衣",
  dateTime: "24 小时前",

```



detail: "冰棍与雪糕绝对不是同一个东西。3 到 5 毛钱的雪糕犹如现在的哈根达斯,而 5 分 1 毛的 冰棍儿就像现在的老冰棒。时过境迁,当年的老冰棍也随着童年的记忆消失不见踪影。记得小时候,每当傍晚时分,总有一个老人推着一辆小车,小车的后架上放着一个大大的白色泡沫盒子。老人一边推着车,一边叫喊着:雪糕、冰棍...",

postId: 1,

music: {

url: "http://ws.stream.qqmusic.qq.com/C100001Dc80Z3qPj2Z.m4a?fromtag=38",

title: "罗大佑 恋曲 1980",

coverImg:

"http://y.gtimg.cn/music/photo_new/T002R150x150M000003cWU1M2qNwxZ.jpg?max_age=2592000",

},

collectionStatus: true,

upStatus: false,

upNum: 11,

comments: []

},

{

date: "Jan 9 2017",

title: "从童年呼啸而过的火车",

postImg: "/images/post/post-5.jpg",

avatar: "/images/avatar/avatar-1.png",

content: "小时候,家的后面有一条铁路。听说从南方北上的火车都必须经过这条铁路。火车大多在晚上经过,但也不定是只有在夜深人静的时候,火车的声音才能从远方传来...",

readingNum: 96,

collectionNum: 7,

commentNum: 4,

author: "林白衣",

dateTime: "24 小时前",

detail: "小时候,家的后面有一条铁路。听说从南方北上的火车都必须经过这条铁路。火车大多在晚上经过,可呜呜的汽笛声往往却被淹没在傍晚小院儿里散步的人群声中。只有在夜深人静的时候,火车的声音才能清晰地从远处飘过来。虽然日日听见火车的汽笛声,可说也奇怪,我竟从来不知道铁路在哪里。在每个夏日午后,我都会有一种去找寻找铁路的冲动,去看看这条铁路究竟是从哪里来,又将通向哪里去",

postId: 2,

music: {

url: "http://ws.stream.qqmusic.qq.com/C100004VybkS2SpZVL.m4a?fromtag=38",

title: "齐秦 原来的我",

coverImg:

"http://y.gtimg.cn/music/photo_new/T002R150x150M000003ZvAeK2PgA4Y.jpg?max_age=2592000"

},

collectionStatus: true,

upStatus: true,

upNum: 22,

comments: [

```

    {
      username: '青石',
      avatar: '/images/avatar/avatar-3.png',
      create_time: '1484723344',
      content: {
        txt: '那一年的毕业季，我们挥挥手，来不及说再见，就踏上了远行的火车。',
        img: ["/images/comment/train-1.jpg", "/images/comment/train-2.jpg",
"/images/comment/train-3.jpg"],
        audio: null
      }
    }, {
      username: '水清',
      avatar: '/images/avatar/avatar-2.png',
      create_time: '1481018319',
      content: {
        txt: '夏日的蝉鸣与夜晚的火车，时常会在未来无数的日子里不断地在我耳边响起，
难以忘怀',
        img: [],
        audio: null,
      }
    },
  ],
  {
    username: '赤墨',
    avatar: '/images/avatar/avatar-1.png',
    create_time: '1484496000',
    content: {
      txt: '时光荏苒，自然的吞噬，让太多的老火车站也消失得无影无踪',
      img: ["/images/comment/train-4.jpg"],
      audio: null,
    }
  },
  {
    username: '林白',
    avatar: '/images/avatar/avatar-4.png',
    create_time: '1484582400',
    content: {
      txt: "",
      img: [],
      audio: {url:"http://123",timeLen:8},
    }
  }
]
},

```




```

{
    date: "Jan 29 2017",
    title: "记忆里的春节",
    postImg: "/images/post/post-1.jpg",
    avatar: "/images/avatar/avatar-3.png",
    content: "年少时，有几样东西，是春节里必不可少的：烟花、新衣、凉菜、压岁钱、饺子。年分大小年，有的地方是腊月二十三过小年，而有的地方是腊月二十四...",
    readingNum: 56,
    collectionNum: 6,
    commentNum: 0,
    author: "林白衣",
    dateTime: "24 小时前",
    detail: "年少时，有几样东西，是春节里必不可少的：烟花、新衣、凉菜、压岁钱、饺子。年分大小年，有的地方是腊月二十三过小年，而有的地方是腊月二十四。童年的春节都是在小县城里度过，那时候的冬天还很冷，池塘的水会结冰，房屋上总是倒挂着一条条的冰凌，菜地里的白菜被厚厚的积雪覆盖着，只露出一小撮白绿相间的菜头，而茎部，竟然像是没有了一般...",
    postId: 3,
    music: {
        url: "http://ws.stream.qqmusic.qq.com/C100003XYcCu3IZKLc.m4a?fromtag=38",
        title: "老狼 虎口脱险",
        coverImg: "http://y.gtimg.cn/music/photo_new/T002R150x150M000002sNbWp3royJG.jpg?max_age=2592000"
    },
    collectionStatus: false,
    upStatus: false,
    upNum: 9,
    comments: []
},
{
    date: "Sep 22 2016",
    title: "换个角度，再来看看微信小程序的开发与发展",
    postImg: "/images/post/post-2.jpg",
    avatar: "/images/avatar/avatar-2.png",
    content: "前段时间看完了雨果奖中短篇获奖小说《北京折叠》。很有意思的是，张小龙最近也要把应用折叠到微信里，这些应用被他称为：小程序...",
    readingNum: 0,
    collectionNum: 0,
    commentNum: 0,
    author: "林白衣",
    dateTime: "24 小时前",
    detail: "我们先举个例子来直观感受下小程序和 App 有什么不同。大家都用过支付宝，在其内部包含着很多小的服务：手机充值、城市服务、生活缴费、信用卡还款、加油服务，吧啦吧啦一大堆服务。这些细小的、功能单一的服务放在支付宝这个超级 App 里，你并不觉得有什么问题，而且用起来也很方便。那

```



如果这些小的应用都单独拿出来，成为一个独立的 App”，

```

    postId: 4,
    music: {
      url: "http://ws.stream.qqmusic.qq.com/C100003z8cOo0Bs7zP.m4a?fromtag=38",
      title: "吴奇隆 祝你一路顺风",
      coverImg: "http://y.gtimg.cn/music/photo_new/T002R150x150M000000n6a7p2HIPqU.jpg?
max_age=2592000"
    },
    collectionStatus: false,
    upStatus: true,
    upNum: 9,
    comments: []
  },
  {
    date: "Jan 29 2017",
    title: "2017 微信公开课 Pro",
    postImg: "/images/post/post-3.jpg",
    avatar: "/images/avatar/avatar-4.png",
    content: "在今天举行的 2017 微信公开课 PRO 版上，微信事业群总裁张小龙宣布，微信“小程序”
将于 1 月 9 日正式上线。”，
    readingNum: 32,
    collectionNum: 2,
    commentNum: 0,
    author: "林白衣",
    dateTime: "24 小时前",
    detail: "在今天举行的 2017 微信公开课 PRO 版上，微信宣布，微信“小程序”将于 1 月 9 日正式
上线，公布了几乎完整的小程序生态模式：微信里没有小程序入口、没有应用市场，分发模式几乎沿用公众
号的模式，去中心化，限制搜索的能力，大多数小程序不能支持模糊搜索，必须输入完整的小程序名称...”，
    postId: 5,
    music: {
      url: "http://ws.stream.qqmusic.qq.com/C100004K7D5E2xhv9v.m4a?fromtag=38",
      title: "杨千嬅 再见二丁目(live)",
      coverImg: "http://y.gtimg.cn/music/photo_new/T002R150x150M000000ptb8p0ZIxDP.jpg?
max_age=2592000",
    },
    collectionStatus: false,
    upStatus: false,
    upNum: 2,
    comments: []
  },
]

```



```
module.exports = {  
  postList: postList  
}
```

以上是完整的 data.js 数据文件。有部分数据目前我们还没有用到，比如 music 数据、文章 id 号、文字详情数据、点赞、评论等。但在后面的章节中，我们将使用以上的诸多数据。开发者可自行修改文字、图片、音乐内容，但需要保持数据结构不变。

再次提醒开发者，更新 data.js 文件后，需要主动在开发工具中清除数据缓存。

也可以访问笔者的微信公众号获取该文件的下载地址。



第 6 章

文章详情页面

本章主要完成文章详情页面的编码工作。在本章中，我们将学习到不同页面间参数的传递技巧、页面跳转的方法、动态设置导航栏标题等知识。

除此之外，本章还介绍了我们经常遇到的一个问题，如何解决元素的垂直居中。

6.1 跳转到文章详情页面

首先新建文章详情页面。在 app.json 的 pages 数组下新增页面路径：

```
"pages/post/post-detail/post-detail"
```

保存后，开发工具会自动生成 post-detail 页面的 4 个文件。首先要实现的是从 post 文章页面通过点击跳转到 post-detail 详情页面。

在 post.wxml 中的 block 代码块里注册一个事件。

代码清单 6-1 注册跳转文章详情页面事件

post.wxml

```
<block wx:for="{{postList}}" wx:for-item="item" wx:for-index="idx">
  <template catchtap="onTapToDetail" is="postItemTpl" data="{{...item}}" />
</block>
```

以上代码仅仅在 template 上增加了一个 catchtap。接着，在 post.js 中编写这个事件的响应函数 onTapToDetail。

代码清单 6-2 跳转到文章详情页面

post.js

```
import { DBPost } from '../db/DBPost.js';
```

```
Page({
  data: {
  },
  onLoad: function () {
    var dbPost = new DBPost();
    this.setData({
      postList: dbPost.getAllPostData()
    });
  },

  onTapToDetail(event){
    wx.navigateTo({
      url: 'post-detail/post-detail',
    })
  }
})
```

添加完 onTapToDetail 函数后，保存运行，并在文章列表页面点击任意一篇文章。没有任何反应，页面也没有跳转。为什么会这样？



6.2 不要在 template 上注册事件

看起来似乎是事件函数并没有响应。此时我们可以在【Wxml】面板中看一下现在页面的骨架结构，如图 6-1 所示。

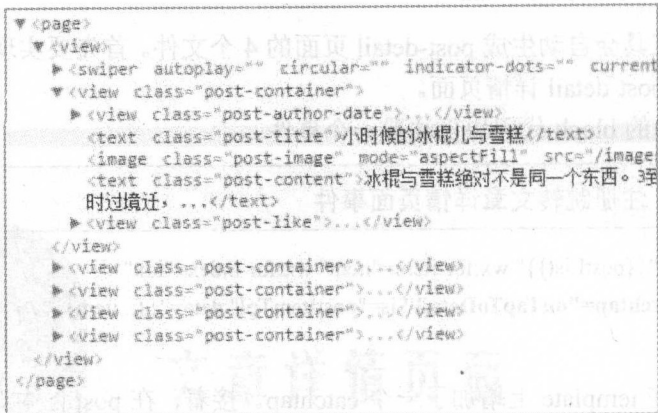


图 6-1 目前 post 页面的骨架结构

是的，注册事件的 template 消失了。在前面章节中我们讲过，template 标签仅仅是一个占位符，在编译后会被 template 的模板内容替换。所以，在 template 上注册事件是无效的。

那么在 block 标签上注册可以吗？来试试，将 catchtap 的事件注册到 block 标签上。

代码清单 6-3 在 block 上注册事件

post.wxml

```
<block catchtap="onTapToDetail" wx:for="{{postList}}" wx:for-item="item" wx:for-index="idx">
  <template is="postItemTpl" data="{{...item}}" />
</block>
```

同样不可以，因为 block 也会在编译后“消失”。

那么，我们只有在 template 的外部增加一个 view，将 template 给包裹起来，并将 catchtap 事件注册到 view 组件上。

代码清单 6-4 增加 view 容器包裹 template 模板

post.wxml

```
<block wx:for="{{postList}}" wx:for-item="item" wx:for-index="idx">
  <view catchtap="onTapToDetail">
    <template is="postItemTpl" data="{{...item}}" />
  </view>
</block>
```

保存后，点击 5 篇文章中的任意一篇，就可以正确地跳转到 post-detail 文章详情页面了。



6.3 页面间传递参数的 3 种方式

上一小节中，我们实现了从文章列表页面跳转到文章详情页面。要正确展示文章详情页面的内容，首先需要将文章的 id 号由 post 页面传递到 post-detail 页面，这样，post-detail 页面才能知晓它要显示哪篇文章。

这涉及页面间的参数传递与通信。目前，在 MINA 框架中有以下几种参数传递方式：

- (1) 使用全局变量（关于全局变量，本书后面的章节中会讲到）。
- (2) 使用缓存。
- (3) 通过页面导航 url 的 query 参数传递。

基本上参数的传递只有以上 3 种方式，其他的比如像事件信号的传参方式（一个页面 emit 发送信号，一个页面 on 监听信号，这种在 AngularJS 里很常见的传参方式，小程序是不能天然支持的），其实都是这些基本思路的变种。

全局变量我们将在后面讲到，至于缓存的传参方式，开发者在学习完缓存后应该很容易想到。其实，使用缓存数据库存储 data.js 初始化数据，又在 post.js 中读取缓存数据，这其实就是页面间的参数传递。仔细想想，是不是这样？

(1)、(2) 两种都涉及全局变量，笔者个人不推荐这种污染全局的传参方式，而且我们的需求仅仅是两个页面间传递参数，完全不需要干扰全局。所以，选用方法 3 来做页面间的参数传递。

6.3.1 组件的自定义属性

再来整理下思路。要将 postId 由 post 页面传递到 post-detail 页面，首先需要在 post.js 中获取到 postId，随后再将 postId 附加到代码清单 6-2 中的 wx.navigateTo 的 url 中。

要想在 post.js 中获取到 postId，就必须知道当前点击的文章是哪一篇文章。我们首先将 postId 绑定到每一篇文章的 wxml 中，使 postId 成为文章 wxml 的一个属性。注意，postId 已在 5.13 小节中加入到 data.js 文件里。

绑定 postId 的方法很简单，就如同绑定文章的 date、title 等属性一样。post.js 文件的代码无须任何改动，只需要对 post.wxml 文件做一下改动即可。

代码清单 6-5 绑定 postId

post.wxml

```
<block wx:for="{{postList}}" wx:for-item="item" wx:for-index="idx">
  <view catchtap="onTapToDetail" data-post-id="{{item.postId}}">
    <template is="postItemTpl" data="{{...item}}" />
  </view>
</block>
```

以上代码中，我们在 view 里增加了一个属性 data-post-id="{{item.postId}}"，先来看看以

上代码有什么效果。保存并运行代码后，打开调试中的【Wxml】这个面板，文章页面的骨架如图 6-2 所示。

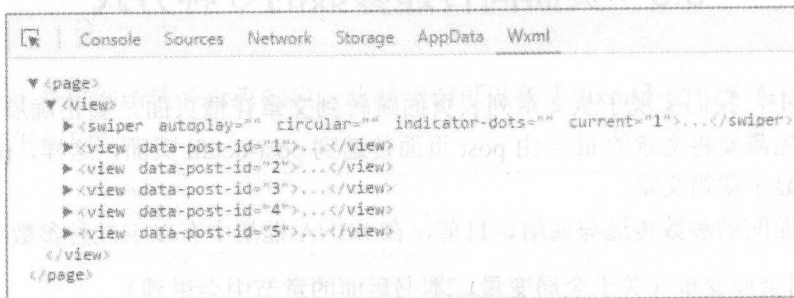


图 6-2 将 postId 绑定在单篇文章的 view 容器上

从图 6-2 中可以很明显地看到，每篇文章的 id 号都被绑定在了该文章的 view 容器上，剩下的工作就是，如何在 post.js 中获取当前点击的文章的 id 号。

6.3.2 通过 dataset 获取组件自定义属性

修改 post.js 文件中的 onTapToDetail 函数如下：

代码清单 6-6 获取 data-post-id

post.js

```
onTapToDetail(event){
  var postId = event.currentTarget.dataset.postId;
  console.log(postId);
  wx.navigateTo({
    url: 'post-detail/post-detail?id=' + postId,
  })
}
```

上述代码中，我们通过 event.currentTarget.dataset.postId 这段代码成功地拿到了当前文章的 postId。

event 事件对象是由 MINA 框架在调用 onTapToDetail 函数时传递的参数。在 event 事件对象中，有一个 currentTarget 代表事件绑定的当前组件。

重点是 dataset 对象，dataset 对象里包含当前组件中所有属性名以 data-开头的自定义属性值。我们在代码清单 6-5 中的 view 上绑定了 data-post-id，所以通过 dataset.postId 将可以拿到当前组件的 postId。

组件自定义属性名有以下规则：

- 必须以 data-开头。
- 多个单词由连字符“-”链接。
- 单词中最好不要有大写字母，如果有大写字母，除单词第一个字母外，其余大写字母将被转化成小写。
- 在 js 中获取自定义属性值时，多个单词将被转化驼峰命名。



看起来很复杂，但举几个例子就非常清楚了，如表 6-1 所示。

表6-1 组件的属性定义

组件的自定义数据	dataset 中的变量名
data-post	dataset.post
data-post-id	dataset.postId
data-pOST-ID	dataset.postId
data-postId	dataset.postid

在获取到 postId 后，我们将 postId 附加在导航 Url 的 query 参数中：

```
url: 'post-detail/post-detail?id=' + postId
```

6.3.3 获取页面参数值

再来看看如何在 post-detail 页面中获取 postId，在 post-detail.js 文件中添加以下代码：

代码清单 6-7 获取页面参数

post-detail.js

```
Page({
  data: {},
  onLoad: function(options){
    var postId = options.id;
  },
})
```

接受 post 页面传递参数的方法是通过 post-detail 页面 onLoad 函数里的 options 参数来获取。options 参数是由框架传递的。

注意，这里 options.id 中的“id”必须同代码清单 6-6 中 navigateTo 中 url 的 query 参数名称保持一致。比如，在 query 参数中使用的是 name=postId，则这里要相应地使用 options.id。

6.4 编译时设置初始化页面及参数

现在我们主要的代码编辑工作集中在 post-detail 这个页面。但每当保存刷新小程序后，项目都将从欢迎页面启动。我们不得不依次点击启动页面、文章列表，才能进入到文章详情页面以预览文章详情的效果，这相当的麻烦。

当然可以按照 4.1 小节中的方法，将 post-detail 页面设置为 app.json 中 pages 数组的第一项，但 post-detail 还需要传入一个 postId 的参数才可以正确地运行，且手动来回更改 pages 数组非常麻烦。

小程序在 122100 版本中增加了编译选项，在此版本中，官方提供了一个“自定义编译”功能，可用于定义小程序的启动页面，如图 6-3 所示。

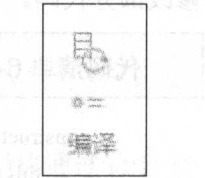


图 6-3 编译选项

122100 版本后，“编译”这里变成了两个选项，请仔细查看。上面一个图标是默认编译，下面的图标是自定义编译，点击下面的图片将弹出如图 6-4 所示的界面。

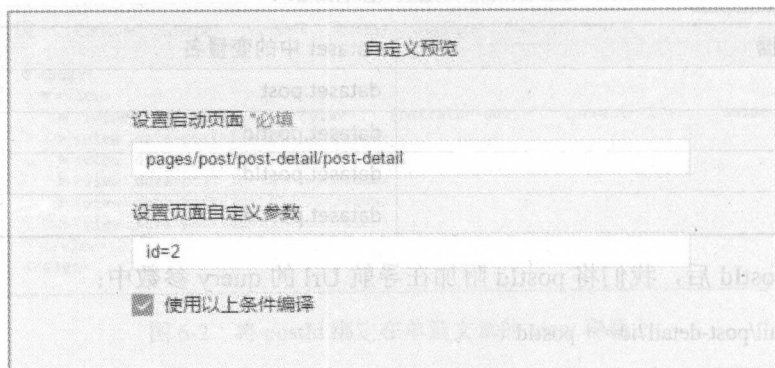


图 6-4 自定义编译选项

启动页面一栏中将路径设置为 post-detail 页面的启动路径：

pages/post/post-detail/post-detail

页面自定义参数类似于 url 中的 query 参数，设置 id=2 将可以在页面中通过 onLoad 函数中的 options.id 获取到这个值。

设置完成后，必须勾选【使用以上条件编译】这个选项。

保存刷新页面，项目将直接进入 post-detail 页面，不再出现 welcome 启动页。同时，我们发现，以这种方式打开的 post-detail 页面无法再返回到 post 页面，因为不是通过 post 页面导航到 post-detail 页面的。

可以通过“&”连接多个页面参数，比如 id=2&name="MR.L"，同时将两个参数传递到 post-detail 页面中。

有了以上功能，就可以非常方便地调试 post-detail 页面。

如果想恢复默认的启动页，只需去掉【使用以上条件编译】选项即可。

6.5 读取文章详情数据

现在，我们已经在文章详情页面中拿到了文章的 postId，接下来需要根据这个 postId 去缓存数据库中读取文章详细数据，并将数据用于构建文章详情页面。

所有对于缓存数据库的操作，我们都会放在 DBPost 这个对象中，在 DBPost.js 中增加和修改部分代码。

代码清单 6-8 新增 getPostItemById 方法

DBPost.js

```
constructor(postId) {
  this.storageKeyName='postList';
  this.postId=postId;
}
```



```
//获取指定 id 号的文章数据
getPostItemById() {
    var postsData = this.getAllPostData();
    var len = postsData.length;
    for (var i = 0; i < len; i++) {
        if (postsData[i].postId === this.postId) {
            return {
                // 当前文章在缓存数据库数组中的序号
                index: i,
                data: postsData[i]
            }
        }
    }
}
```

注意，以上代码只标注出了相关修改和增加代码，并非全部代码。首先修改 `constructor` 构造函数，增加一个构造参数 `postId`，并将 `postId` 保存到 `this` 变量中。

接着增加一个方法 `getPostItemById` 用于获取指定 `id` 号的文章数据。

`DBPost` 修改完毕后，我们尝试在 `post-detail.js` 中获取指定 `id` 号的文章数据，并使用 `this.setData` 绑定该数据。

代码清单 6-9 获取指定 `id` 号的文章数据

post-detail.js

```
onLoad: function (options) {
    var postId = options.id;
    this.dbPost = new DBPost(postId);
    this.postData = this.dbPost.getPostItemById().data;
    this.setData({
        post: this.postData
    })
},
```

注意上述代码中，在使用 `new` 实例化 `DBPost` 后，将 `dbPost` 这个对象保存在了变量 `this` 中，这样以后如果要再次使用 `DBPost`，则不需要再重新实例化这个对象，只需要使用 `this.dbPost` 即可引用这个对象。

6.6 文章 `id` 号的数据流向图

我们来梳理一下，`post-detail.js` 是如何从初始化数据中拿到文章 `id` 号，并最终通过 `id` 号来获取到文章详情数据的，参见图 6-5 所示。



文前 id 号的数据流向

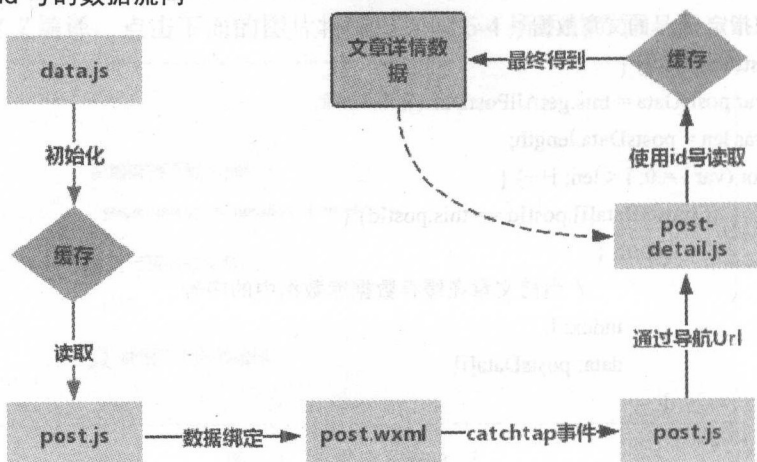


图 6-5 文章 id 号流向图

文章 id 号最初是存在于 data.js 中的，通过一系列的事件操作，它最终会被传递到 post-detail.js 中。一旦 post-detail.js 拿到文章的 id 号，该页面就可以根据 id 号来获取文章详情数据了。

6.7 编写文章详情页面

在代码清单 6-9 中，我们获取了文章的 postId，并通过 DBPost 查询到了该文章的相关数据，随后我们用 this.setData 函数做了文章数据的数据绑定。下面，我们来编写文章详情页面的骨架和样式。

在 post-detail.wxml 中加入以下页面骨架代码：

代码清单 6-10 编写详情页面骨架

post-detail.wxml

```

<view class="container">
  <image class="head-image" src="{{post.postImg}}"/></image>
  <text class="title">{{post.title}}</text>
  <view class="author-date">
    <view class="author-box">
      <image class="avatar" src="{{post.avatar}}"/></image>
      <text class="author">{{post.author}}</text>
    </view>
    <text class="date">{{post.dateTime}}</text>
  </view>
  <text class="detail">{{post.detail}}</text>
</view>

```



还是要注意代码中`{{}}`中的数据绑定语法一定要正确，否则无法读取数据。保存并运行代码，`post-detail` 页面将显示这些文章数据。

但整个页面的样式是错乱的，因为还没有编写 `post-detail` 页面的 `wxss` 文件。

代码清单 6-11 编写详情页面的样式

post-detail.wxss

```
.container {  
  display: flex;  
  flex-direction: column;  
}  
  
.head-image {  
  width: 750rpx;  
  height: 460rpx;  
}  
  
.title {  
  font-size: 20px;  
  margin: 30rpx;  
  letter-spacing: 2px;  
  color: #4b556c;  
}  
  
.author-date {  
  display: flex;  
  flex-direction: row;  
  margin-top: 15rpx;  
  margin-left: 30rpx;  
  align-items: center;  
  justify-content: space-between;  
  font-size: 13px;  
}  
  
.author-box {  
  display: flex;  
  flex-direction: row;  
  align-items: center;  
}  
  
.avatar {  
  height: 50rpx;  
  width: 50rpx;  
}
```



```
.author {  
  font-weight: 300;  
  margin-left: 20rpx;  
  color: #666;  
}
```

```
.date {  
  color: #919191;  
  margin-right: 38rpx;  
}
```

```
.detail {  
  color: #666;  
  margin: 40rpx 22rpx 0;  
  line-height: 44rpx;  
  letter-spacing: 1px;  
  font-size: 14px;  
}
```

保存并刷新页面后，文章详情页面将正确地显示出来，如图 6-6 所示。



图 6-6 文章详情页面

6.8 垂直居中问题的经典解决方法

我们在编写 CSS 时，很多时候都会面临如何将两个元素垂直居中对齐的问题。比如在代码清单 6-11 中如何将作者名称 (author) 和作者头像 (avatar) 垂直居中对齐。

我们在 3.4 小节中学习了 flex，这里就来看看如何使用 flex 解决这个问题。

代码清单 6-12 文字和图片垂直居中对齐

post-detail.wxss

```
.author-box {  
  display: flex;  
  flex-direction: row;  
  align-items: center;  
}
```

```
.avatar {  
  height: 50rpx;  
  width: 50rpx;  
}
```

```
.author {  
  font-weight: 300;  
  margin-left: 20rpx;  
  color: #666;  
}
```

以上代码摘自 post-detail.wxss。解决思路如下：将 avatar 和 author 用一个容器包裹起来 (author-box)，使用 display: flex 将该容器设置为 flex 盒子模型，使用 flex-direction: row 指定 flex 的方向为 row。

关键的代码是 align-items: center，这将使 flex 盒子里的元素在交叉轴方向上居中。在本例中主轴是水平方向（因为设置了 flex-direction 为 row），所以交叉轴是垂直方向，align-items: center 将控制垂直方向居中。关于 flex 及轴的概念已经在 3.4 小节中详细讲解过，开发者可自行回顾一下关于主轴和交叉轴的概念。

开发者可以对比一下，welcome 页面中是如何使头像、文字和按钮这 3 个元素水平居中的。welcome 页面中设置了 flex-direction: column，所以主轴是垂直方向，align-items: center 将控制水平方向上的居中。

小程序对于 flex 的支持相当完善，建议多使用 flex 进行元素布局。

6.9 动态设置导航栏标题

本小节，我们来学习如何在页面导航栏中设置标题。导航栏是页面最顶部的一块区域，如图 6-7 所示。



图 6-7 导航栏的位置位于页面最顶部

它现在光秃秃的，没有任何文字。这里，我们使用两种方法分别设置 post 页面和 post-detail 页面的导航栏文字。

6.9.1 使用配置文件配置导航栏标题

第一种方法是使用 app.json 或者页面的 json 文件来配置导航栏标题。如果是在 app.json 中进行配置，则它是全局行为，项目所有的页面将显示同一个标题；而如果是在页面的 json 文件中配置标题，则只会影响当前配置页面。

我们之前在 app.json 的 window 属性中通过设置 navigationBarBackgroundColor 实现了指定导航栏的颜色。window 还有以下 2 个属性用于配置导航栏文本：

- navigationBarTextStyle 指定导航栏标题文字的颜色，只支持 black/white，默认值为 black。
- navigationBarTitleText 指定导航栏标题文字。

在 app.json 中对 window 属性增加以上两个配置项，代码如下：

代码清单 6-13 配置导航栏文字

app.json

```
"window": {  
  "navigationBarBackgroundColor": "#4A6141",  
  "navigationBarTextStyle": "white",  
  "navigationBarTitleText": "文 字"  
}
```

保存运行后可以看到，所有页面的导航栏都增加了“文字”这两个字，它的颜色为白色。这并不是我们想要的，我们希望不同页面显示不同的导航栏标题。下面来解决这个问题。

我们之前讲过，window 这个配置项既可以在 app.json 中配置，也可以在 window 中配置（其他配置项只能在 app.json 中配置）。

删除代码清单 6-13 中的 `navigationBarTextStyle` 和 `navigationBarTitleText`，在 `post.json` 文件中加入以下代码：

代码清单 6-14 在页面中配置导航栏文字

post.json

```
{
  "navigationBarTextStyle": "white",
  "navigationBarTitleText": "文 字"
}
```

这样，导航栏文字就只会在 `post` 页面中出现了。

6.9.2 使用 `wx.setNavigationBarTitle(OBJECT)` 设置导航条

在某些情况下，我们希望导航栏的文字可以根据页面内容的不同而有所变化。比如在文章详情页面中，我们希望导航栏可以实时显示当前文章的标题，不同的文章显示不同的标题文字。来看看如何实现这个功能。

小程序提供了 `wx.setNavigationBarTitle(OBJECT)` 来动态设置导航栏标题。小程序官方文档中指出，页面的导航栏标题必须在页面生命周期的 `onReady` 之后来设置，否则无效。原文如下：

对界面的设置如 `wx.setNavigationBarTitle` 请在 `onReady` 之后设置。

我们遵照官方文档的说明，在 `post-detail.js` 中加入以下代码：

代码清单 6-15 动态设置导航栏文字

post-detail.js

```
onReady:function(){
  wx.setNavigationBarTitle({
    title: this.postData.title
  })
}
```

按照文档的描述，我们在页面生命周期函数 `onReady` 中调用了 `wx.setNavigationBarTitle(object)` 方法。它接收一个 `object` 参数，其中 `title` 属性被设置为当前文章的标题。

保存运行代码，发现页面的导航栏文字变成了文章的标题。

这里需要指出，在 122100 版本之前，`setNavigationBarTitle(object)` 方法确实只能在页面的 `onReady` 函数里设置。如果尝试在页面的 `onLoad`、`onShow` 函数里调用 `setNavigationBarTitle(object)` 方法，文章的标题将出现一闪而过的情况。这种情况是符合官方文档说明的：“对界面的设置如 `wx.setNavigationBarTitle` 请在 `onReady` 之后设置”。因为 `onReady` 在 `onShow` 发生之后才触发，`onShow` 将标题设置完毕后，`onReady` 会重新渲染页面，并覆盖导航栏的标题，这就是我们说的“一闪而过”的情况。

但在最新的 130400 版本里，无论是在页面的 `onLoad` 或者 `onShow` 函数中调用 `setNavigationBarTitle(object)` 方法，都可以成功地设置导航栏标题，并不会出现一闪而过的情况。

但无论如何，还是建议开发者按照官方文档所描述的，在 `onReady` 函数里进行界面的设置操作，以免官方在未来再次改动底层的运行机制时造成代码无法运行。



第 7 章

收藏、评论、点赞与计数功能

本章内容有一定的难度，但其中的技巧和知识还是很丰富的。本章通过编写几乎所有内容型应用都会附带的“评论”“点赞”“阅读计数”“收藏”等功能，来学习使用小程序的交互反馈组件、缓存的应用、图片选择和预览、屏蔽关键字、录音、拍照以及播放录音等功能。

7.1 收藏、评论、点赞、计数功能准备工作

接下来我们将要连续实现 4 个非常有意思的功能，这些功能在内容型应用中是非常常见的，分别是收藏、点赞、评论和计数。

我们先来编写收藏、评论和点赞的功能按钮。阅读计数是一项被动功能，无须用户有意识地主动触发。在 post-detail.wxml 中新增一段工具栏代码。

代码清单 7-1 编写 3 个功能的功能按钮

post-detail.wxml

```
<view class="container">
  <image class="head-image" src="{{post.postImg}}"></image>
  <text class="title">{{post.title}}</text>
  <view class="author-date">
    <view class="author-box">
      <image class="avatar" src="{{post.avatar}}"></image>
      <text class="author">{{post.author}}</text>
    </view>
    <text class="date">{{post.dateTime}}</text>
  </view>
  <text class="detail">{{post.detail}}</text>
</view>
<view class="tool">
  <view class="tool-item" catchtap="onUpTap" data-post-id="{{post.postId}}">
    <image src="/images/icon/wx_app_like.png" />
    <text>{{post.upNum}}</text>
  </view>
  <view class="tool-item comment" catchtap="onCommentTap" data-post-id="{{post.postId}}">
    <image src="/images/icon/wx_app_message.png"></image>
    <text>{{post.commentNum}}</text>
  </view>
  <view class="tool-item" catchtap="onCollectionTap" data-post-id="{{post.postId}}">
    <image src="/images/icon/wx_app_collect.png" />
    <text>{{post.collectionNum}}</text>
  </view>
</view>
```

黑色加粗部分为新增代码。

在 post-detail.wxml 页面的 container 中添加了一段 `<view class="tool">` 的相关代码。该代码实现了收藏、评论和点赞 3 个功能按钮。每个功能按钮都绑定了对应的点击事件，注意 view 组件上的 catchtap 属性。除此之外，我们还在每个功能按钮上使用 data-post-id 绑定了当前文章的 id 号。

upNum、commentNum 和 collectionNum 等数据已经在 5.13 小节中全部添加到了 data.js 文件中。

接着编写 3 个功能按钮的样式。

代码清单 7-2 编写 3 个功能按钮的样式

post-detail.wxss

```
.tool{
  height: 64rpx;
  text-align: center;
  line-height: 64rpx;
  margin: 20rpx 28rpx 20rpx 0;
}
.tool-item{
  display: inline-block;
  vertical-align: top;
  margin-right: 30rpx;
}

.tool-item image{
  height: 30rpx;
  width:30rpx;
  vertical-align: -3px;
  margin-right: 10rpx;
}

.comment image{
  transform: scale(.85);
}
```

保存刷新后，3 个功能按钮将出现在 post-detail 页面的正下方，如图 7-1 所示。

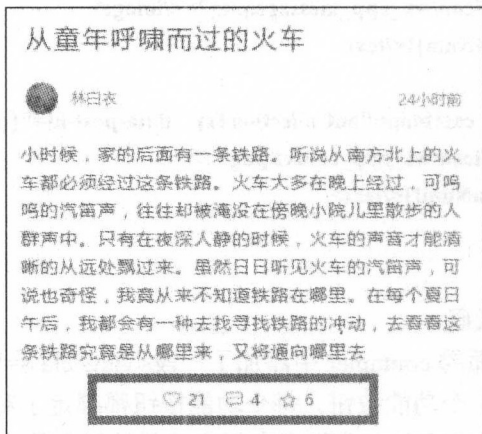


图 7-1 3 个功能按钮的样式和位置

7.2 文章收藏功能

我们首先来实现文章收藏功能。文章收藏功能需要记录两个变量值：

- 自己是否收藏了文章。如果自己收藏了，那么需要将收藏的图片更换为已收藏。
- 所有用户收藏文章的总数量。需要注意的是，由于我们的数据库只在本地，无法多次收藏同一篇文章，所以收藏数量永远只能在初始数量的基础上+1 或者-1，分别对应取消收藏和点击收藏两种状态。但在真实的项目中，这个收藏数量却是要受到所有用户取消、收藏文章动作影响的。同样的情况也会出现在“文章点赞”这个功能里。

当页面从 post 跳转到 post-detail 时，我们就需要知道该文章是否已被用户收藏。在 data.js 中，我们使用 collectionStatus 这个属性表示文章是否已被收藏，这个变量的类型是 Boolean。那么如何根据 collectionStatus 这个变量的取值来动态切换收藏图标呢？

熟悉传统 Web 开发的读者很容易想到用 jQuery 获取 image 标签，再动态地设置 image 的 src 属性。再次强调，小程序没有 dom，一切都是数据绑定，请抛弃 dom 的思维方式。

7.2.1 条件渲染：wx:if 与 wx:else

collectionStatus 只有两种取值：true 或者 false。我们需要做的是，当 collectionStatus 为 false 时，显示图 7-2 未收藏状态的图标，而当 collectionStatus 为 true 时，显示图 7-3 收藏状态的图标。



图 7-2 未收藏状态图标



图 7-3 收藏状态图标

以上需求是不是就是编程中非常经典的 if else？如果 wxml 组件也像 js 代码一样有 if else 就可以解决动态显示收藏图片的问题。下面来看看如何实现这个功能。

小程序提供了 wx:if 与 wx:else 来实现条件渲染。当变量为 true 时，执行 wx:if，否则将执行 wx:else。修改收藏按钮的 wxml 代码如下：

代码清单 7-3 条件渲染

post-detail.wxml

```
<view class="tool-item" catchtap="onCollectionTap" data-post-id="{{post.postId}}">
  <image wx:if="{{post.collectionStatus}}" src="/images/icon/wx_app_collected.png" />
  <image wx:else src="/images/icon/wx_app_collect.png" />
  <text>{{post.collectionNum}}</text>
</view>
```

上述代码中我们添加了两个 image 组件，分别是收藏和未收藏图片。这两个 image 组件各有一个 wx:if 和 wx:else 属性。当 post.collectionStatus 为 true 时将显示 wx_app_collected.png



图片，而当 `post.collectionStatus` 为 `false` 时将显示 `wx_app_collect.png` 图片。

由于我们已经在 `data.js` 文件中将部分文章的收藏状态设置为 `true`，因此保存并运行项目，发现所有 `collectionStatus` 为 `true` 的文章，其收藏图片都将显示 `wx_app_collected.png`，如图 7-4 所示。

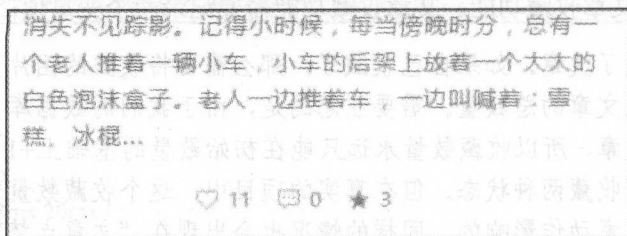


图 7-4 显示 `wx_app_collected.png`

`wx:if` 与 `wx:else` 的条件渲染在小程序中被大量使用，不仅仅被用来做图片的更换，还可以用来控制元素的显示和隐藏。

`wx:if` 可以被单独使用，并不一定要同 `wx:else` 一起使用。

除此之外，条件渲染还可以做多级别的 `if else`，如代码清单 7-4 的示例代码所示。

代码清单 7-4 多级别 `if else`

示例

```
<view wx:if="{{length > 5}}"> 1 </view>
<view wx:elif="{{length > 2}}"> 2 </view>
<view wx:else> 3 </view>
```

如果变量 `length` 的取值大于 5，那么将显示数字 1。

如果变量 `length` 的取值大于 2 且小于等于 5，那么将显示数字 2。

以上条件都不满足，就显示数字 3。

你还可以添加更多的 `elif` 分支，以实现更多级别的条件判断。

7.2.2 实现收藏点击功能

在 7.2.1 小节中，我们仅仅是在 `post-detail` 页面加载时读取了该文章对于当前用户是否为收藏状态，并正确地设置和显示了这个状态。

在这个小节中，我们将实现用户点击图片进行文章的收藏和取消收藏功能。首先我们继续完善 `DBPost` 这个数据库操作类。在 `DBPost` 类中添加一个方法，用以处理文章的收藏操作。

代码清单 7-5 添加处理文章收藏的方法

`DBPost.js`

```
// 收藏文章
collect(){
  return this.updatePostData('collect');
}
```

该方法中调用了 `DBPost` 类的 `updatePostData` 方法，这个方法我们还没有编写。

在 DBPost 类中添加 updatePostData 方法。该方法是处理点赞、评论、收藏、阅读的核心方法。

代码清单 7-6 添加 updatePostData 方法

DBPost.js

```
//更新本地的点赞、评论信息、收藏、阅读量
updatePostData(category) {
    var itemData = this.getPostItemById(),
        postData = itemData.data,
        allPostData = this.getAllPostData();
    switch (category) {
        case 'collect':
            //处理收藏
            if (!postData.collectionStatus) {
                //如果当前状态是未收藏
                postData.collectionNum++;
                postData.collectionStatus = true;
            } else {
                // 如果当前状态是收藏
                postData.collectionNum--;
                postData.collectionStatus = false;
            }
            break;
        default:
            break;
    }
    // 更新缓存数据库
    allPostData[itemData.index] = postData;
    this.execSetStorageSync(allPostData);
    return postData;
}
```

我们目前仅处理 collect 这一种操作，后续我们将继续在代码清单 7-6 的 switch case 中添加评论、阅读数、点赞等处理分支。

这样，DBPost 就具备了处理文章收藏的能力。当用户点击收藏按钮后，在点击事件函数中调用 DBPost 的 collect 方法即可。

处理文章收藏动作的事件函数是 onCollectionTap，这个事件函数已在代码清单 7-1 中被注册在了收藏功能按钮上。我们只需要在 post-detail.js 中编写这个方法即可。

代码清单 7-7 编写 onCollectionTap 方法

post-detail.js

```
onCollectionTap: function (event) {
    //dbPost 对象已在 onLoad 函数里被保存到了 this 变量中，无须再次实例化
```

```

var newData = this.dbPost.collect();

//重新绑定数据。注意，不要将整个 newData 全部作为 setData 的参数，
//应当有选择地更新部分数据
this.setData({
  'post.collectionStatus': newData.collectionStatus,
  'post.collectionNum':newData.collectionNum
})
}

```

7.2.3 交互反馈 wx.showToast

现在，我们已经实现了文章的收藏与取消收藏功能，但收藏功能的体验并不好，用户在收藏和取消收藏后没有任何交互反馈提示。

小程序提供了一些交互反馈 API 来帮助开发者处理交互相关的问题。目前，小程序提供了以下 4 个交互反馈 API：

- wx.showToast
- wx.hideToast
- wx.showModal
- wx.showActionSheet

我们选用 wx.showToast(object)来制作文章收藏功能的交互反馈。

代码清单 7-8 文章收藏功能的交互反馈

post-detail.js

```

// 交互反馈
wx.showToast({
  title: newData.collectionStatus ? "收藏成功" : "取消成功",
  duration: 1000,
  icon: "success",
  mask:true
})

```

其中，object 参数的 title 属性用于设置提醒消息的内容；duration 设置提醒的自动消失时间，最长 10000 毫秒，默认值为 1500 毫秒；icon 可以设置一个小图标，其取值只能是 success 和 loading；mask 指定是否显示透明的蒙层，以防止触摸穿透，默认值为 false。

mask 主要用来防止用户连续点击收藏按钮。开发者可执行尝试将 mask 设置为 true 和 false 时的不同效果：当 mask 为 true 时连续点击收藏图标，图标不会连续做出收藏/取消收藏的响应；当 mask 为 false 时，就会不停地响应用户的点击操作。

wx.showToast 的效果如图 7-5 所示。





图 7-5 wx.showToast 效果

7.3 文章点赞功能

文章点赞功能的实现思路同收藏几乎是一样的。

首先在 DBPost.js 中增加点赞的方法。

代码清单 7-9 添加处理点赞操作的方法

DBPost.js

```
// 点赞或者取消点赞
```

```
up() {
  var data = this.updatePostData('up');
  return data;
}
```

接着在 DBPost 的 updatePostData 方法中处理当 case 为 up 时的情况。下面给出 updatePostData 的全部代码。

代码清单 7-10 增加 case 为 up 时的处理方法

DBPost.js

```
//更新本地的点赞、评论信息、收藏、阅读量
```

```
updatePostData(category) {
  var itemData = this.getItemById(),
  postData = itemData.data,
  allPostData = this.getAllPostData();
  switch (category) {
    case 'collect':
      //处理收藏
      if (!postData.collectionStatus) {
        //如果当前状态是未收藏
```

```

        postData.collectionNum++;
        postData.collectionStatus = true;
    } else {
        // 如果当前状态是收藏
        postData.collectionNum--;
        postData.collectionStatus = false;
    }
    break;
case 'up':
    if (!postData.upStatus) {
        postData.upNum++;
        postData.upStatus = true;
    } else {
        postData.upNum--;
        postData.upStatus = false;
    }
    break;
default:
    break;
}
allPostData[itemData.index] = postData;
this.execSetStorageSync(allPostData);
return postData;
}

```

黑色加粗部分为新增代码。代码清单 7-10 同代码清单 7-6 相比，仅仅增加了 case 为 'up' 时的这段代码。很明显，我们可以看到处理点赞的逻辑同处理收藏时的逻辑几乎一样：改变 upStatus 的状态，并对 upNum 这个计数变量做相应的增减操作。

我们编写完 DBPost 中关于点赞的接口后，接着编写 post-detail.js 和 post-detail.wxml 中关于点赞的相关代码。

代码清单 7-11 编写 onUpTap 方法

post-detail.js

```

onUpTap:function(event){
    var newData = this.dbPost.up();

    this.setData({
        'post.upStatus': newData.upStatus,
        'post.upNum':newData.upNum
    })
}

```

onUpTap 方法响应用户点赞的动作。当用户点击点赞按钮后，onUpTap 方法将调用 DBPost 的 up 方法并将返回的最新数据使用 this.setData 更新。



类似于收藏功能，我们还需要使用条件渲染 `wx:if` 改写 `wxml` 中的点赞按钮。

代码清单 7-12 点赞功能的条件渲染

post-detail.wxml

```
<view class="tool-item" catchtap="onUpTap" data-post-id="{{post.postId}}">
  <image wx:if="{{post.upStatus}}" src="/images/icon/wx_app_liked.png" />
  <image wx:else src="/images/icon/wx_app_like.png" />
  <text>{{post.upNum}}</text>
</view>
```

以上代码将在 `post.upStatus` 为 `true` 时显示 `wx_app_liked.png`，当 `post.upStatus` 为 `false` 时显示 `wx_app_like.png`。

在编写完以上代码后，保存运行项目，点击点赞按钮，图片会不断切换，点赞数也将相应地+1 或者-1。

很多开发者可能还不太习惯使用数据绑定的方式来做样式、状态的切换，但数据绑定的写法确实非常简化、方便。我们只需要在 `js` 中改变各类变量的状态和值，前端组件就会响应我们的操作，动态地做出变化。

7.4 本地缓存的重要性及应用举例

提供本地的 `key&value` 缓存机制是小程序的一大特点，善用本地缓存将可以极大地改善客户端的体验与服务器的性能。

前几个小节中，我们大量地使用了本地缓存来模拟服务器的数据库。这样做一方面是因为我们并没有真实的服务器，必须依靠客户端的缓存能力来记录数据；另一方面是因为即使在真实的项目中我们拥有自己的远程服务器，也依然需要在客户端管理本地缓存。

举个例子，如果我们要实现一个城市列表插件，就必然要获取全国所有城市的信息。全国大概有 600 多个城市，这么大的数据量难道每次打开这个插件都要去服务器取城市数据吗？这些城市的数据相对非常稳定，并不会频繁变化，每次都去服务器加载是对流量和服务器性能的严重消耗。

所以，最好的解决方案就是将城市数据保存在本地缓存中，而不是每次都去服务器请求数据。

在一个高性能的产品中，缓存的重要性是不言而喻的。建议开发者将本地缓存视作一个本地的 `key&value` 数据库，并封装一些类和公共方法，提供给项目中的各个调用方。最好不要让 `getStorage`、`setStorage` 等方法充斥在项目的每一个角落。

Orange Can 项目中的 `DBPost` 类就是一个不错的示例，它实现了对缓存的良好管理，并向调用方提供了一系列可读性非常强的 API。建议开发者参考 `DBPost` 并将这种思路应用到自己的项目中。

7.5 支持文字、图片、拍照、语音上传的文章评论

文章评论不仅可以发表文字，还可以上传图片和语音。评论页面将使用一个全新的 post-comment 页面，它属于 post-detail 的子页面。我们将通过点击评论功能按钮跳转到 post-comment 页面。

首先，在 app.json 的 pages 数组下注册 post-comment 页面。

代码清单 7-13 注册 post-comment 页面

app.json

```
"pages": [  
  "pages/welcome/welcome",  
  "pages/post/post",  
  "pages/post/post-detail/post-detail",  
  "pages/post/post-comment/post-comment"  
],
```

保存后将自动在项目里创建 post-comment 目录以及页面的 4 种类型文件。post-comment 将作为 post-detail 的子页面。在 post.js 中添加以下代码：

代码清单 7-14 onCommentTap 方法

post-detail.js

```
onCommentTap: function (event) {  
  var id = event.currentTarget.dataset.postId;  
  wx.navigateTo({  
    url: '../post-comment/post-comment?id=' + id  
  })  
}
```

以上代码将携带当前文章的 id 号并跳转到 post-comment 页面。

7.6 文章评论页面的实现步骤与思路

构建文章评论页面的思路分为两部分：

- 加载并显示当前文章已存在的评论。
- 实现添加新评论的功能。

这个思路不是做特定功能的思路，而是一种适用于大部分前端功能的通用思路，就像我们在 post-detail 页面中去编写点赞、收藏等功能时一样，先显示点赞和收藏的数量、状态，再考虑实现点赞和收藏的操作功能。



我们来按照以下步骤逐步构建整个文章评论模块的相关页面和功能：

- 在 `post-comment.js` 中获取并绑定文章评论数据。
- 编写 `post-comment` 页面的 `wxml` 和 `wxss` 显示文章评论数据。
- 编写添加新评论的功能。

7.7 获取并绑定文章评论数据

我们在 `data.js` 中《从童年呼啸而过的火车》这篇文章下面模拟了 4 条评论数据（`comments` 数组）。

按照 7.6 节中分析的思路，首先应当从缓存数据库中读取这 4 条数据并将数据绑定到框架中。首先在 `post-comment.js` 中增加以下代码：

代码清单 7-15 获取评论数据

`post-comment.js`

```
import { DBPost } from '../db/DBPost.js';

Page({
  data: {},
  onLoad: function (options) {
    var postId = options.id;
    this.dbPost = new DBPost(postId);
    var comments = this.dbPost.getCommentData();

    // 绑定评论数据
    this.setData({
      comments: comments
    });
  }
})
```

以上代码引入了 `DBPost` 缓存数据库操作类，同时在 `onLoad` 函数里接收由 `post-detail.js` 传递过来的 `postId`。接着实例化 `DBPost`，再调用 `DBPost` 的 `getCommentData` 方法得到评论数据，最后使用 `this.setData` 将评论数据绑定到框架中。

很明显，我们的 `DBPost` 中缺少 `getCommentData` 方法，现在来编写这个方法。在 `DBPost.js` 中新增以下代码：

代码清单 7-16 编写获取文章评论的方法

`DBPost.js`

```
//获取文章的评论数据
getCommentData () {
  var itemData=this.getItemById().data;
```



```
//按时间降序排列评论
itemData.comments.sort(this.compareWithTime);
var len=itemData.comments.length,
    comment;
for(var i=0;i<len;i++) {
    // 将 comment 中的时间戳转换成可阅读格式
    comment=itemData.comments[i];
    comment.create_time=util.getDiffTime(comment.create_time,true);
}
return itemData.comments;
}
```

在 `getCommentData` 这个方法中，我们还调用了 `compareWithTime` 和 `util.getDiffTime` 这两个方法。`compareWithTime` 用于将评论按照时间降序排列，保证最新的评论在最上方；`util.getDiffTime` 将评论的时间戳转化为“多少分钟前，多少小时前，昨天，月日”等格式。下面我们来实现这两个方法。

在 `DBPost.js` 中添加 `compareWithTime` 方法，代码如下：

代码清单 7-17 `compareWithTime` 方法

`DBPost.js`

```
compareWithTime(value1,value2) {
    var flag=parseFloat(value1.create_time) - parseFloat(value2.create_time);
    if(flag<0){
        return 1;
    }else if(flag>0){
        return -1
    }else{
        return 0;
    }
}
```

接着实现 `util.getDiffTime` 方法。这个方法属于公共方法，我们新建一个 `util` 文件，将所有的公共方法都放置到这个 `util` 文件中。

在根目录下新建文件夹 `util`，在 `util` 文件夹下新建 `util.js` 文件并在该文件中新增以下代码：

代码清单 7-18 `getDiffTime` 方法

`util.js`

```
/*
 *根据客户端的时间信息得到发表评论的时间格式
 *多少分钟前，多少小时前，昨天，月日
 * Para :
 * recordTime - {float} 时间戳
 * yearsFlag - {bool} 是否要年份
 */
```



```

function getDiffTime(recordTime,yearsFlag) {
    if (recordTime) {
        recordTime=new Date(parseFloat(recordTime)*1000);
        var minute = 1000 * 60,
            hour = minute * 60,
            day = hour * 24,
            now=new Date(),
            diff = now -recordTime;

        var result = "";
        if (diff < 0) {
            return result;
        }

        var weekR = diff / (7 * day);
        var dayC = diff / day;
        var hourC = diff / hour;
        var minC = diff / minute;
        if (weekR >= 1) {
            var formate='MM-dd hh:mm';
            if(yearsFlag){
                formate='yyyy-MM-dd hh:mm';
            }
            return recordTime.format(formate);
        }
        else if (dayC == 1 ||(hourC <24 && recordTime.getDate()!==now.getDate())) {
            result = '昨天'+recordTime.format('hh:mm');
            return result;
        }
        else if (dayC > 1) {
            var formate='MM-dd hh:mm';
            if(yearsFlag){
                formate='yyyy-MM-dd hh:mm';
            }
            return recordTime.format(formate);
        }
        else if (hourC >= 1) {
            result = parseInt(hourC) + '小时前';
            return result;
        }
        else if (minC >= 1) {
            result = parseInt(minC) + '分钟前';
            return result;
        }
        else {
            result = '刚刚';
        }
    }
}

```

```

        return result;
    }
}
return "";
}

```

上述 getDiffTime 方法中调用了 Date 的 format 方法。继续添加 format 这个方法。需要注意的是, format 方法被添加在了 Date 的原型链上, 这样所有 Date 类型的变量都将自动拥有 format 这个方法。

代码清单 7-19 在 Date 原型链上新增 format 方法

util.js

```

/*
 * 此方法来源于网络
 * 拓展 Date 方法, 得到格式化的日期形式
 * date.format('yyyy-MM-dd'), date.format('yyyy/MM/dd'), date.format('yyyy.MM.dd')
 * date.format('dd.MM.yy'), date.format('yyyy.dd.MM'), date.format('yyyy-MM-dd HH:mm')
 * 使用方法 如下:
 *   var date = new Date();
 *   var todayFormat = date.format('yyyy-MM-dd');
 * Parameters:
 * format - {string} 目标格式 类似('yyyy-MM-dd')
 * Returns - {string} 格式化后的日期
 *
 */

```

```

(function initTimeFormat(){
    Date.prototype.format = function (format) {
        var o = {
            "M+": this.getMonth() + 1, //month
            "d+": this.getDate(), //day
            "h+": this.getHours(), //hour
            "m+": this.getMinutes(), //minute
            "s+": this.getSeconds(), //second
            "q+": Math.floor((this.getMonth() + 3) / 3), //quarter
            "S+": this.getMilliseconds() //millisecond
        }
        if (/([y+]).test(format)) format = format.replace(RegExp.$1,
            (this.getFullYear() + "").substr(4 - RegExp.$1.length));
        for (var k in o) if (new RegExp("(" + k + ")").test(format))
            format = format.replace(RegExp.$1,
                RegExp.$1.length === 1 ? o[k] :
                ("00" + o[k]).substr(("" + o[k]).length));
    }
}

```




```

    return format;
  };
}()

```

不需要深入研究上面这段代码，只需要知道它的作用即可。最后，在 `util.js` 的末尾添加以下代码：

代码清单 7-20 添加 module.exports

`util.js`

```

module.exports = {
  getDiffTime: getDiffTime
}

```

这样 `util.js` 这个模块就编写完成了。为了在 `DBPost.js` 中引用这个模块，我们需要在 `DBPost.js` 中使用 `require` 来引用 `util` 模块。

代码清单 7-21 引用 util 模块

`BPost.js`

```

var util = require('../util/util.js')

```

编写完以上代码后，`DBPost` 的 `getCommentData` 就完成了。现在，在 `post-comment.js` 的 `onLoad` 方法里就可以正确获取文章的评论数据。开发者可自行在代码清单 7-15 中使用 `console.log(comments)` 来验证一下是否能够正确获取到文章评论。需要注意的是，我们仅仅在《从童年呼啸而过的火车》这篇文章里设置了 4 条评论数据，其他文章是没有评论数据的。开发者可根据自身的需求修改 `data.js` 中的初始化数据。注意，修改完 `data.js` 后一定要用开发工具清除一下缓存，并重新运行项目，之后更改才能生效。

图 7-6 是输出的《从童年呼啸而过的火车》这篇文章的 `comments` 变量。

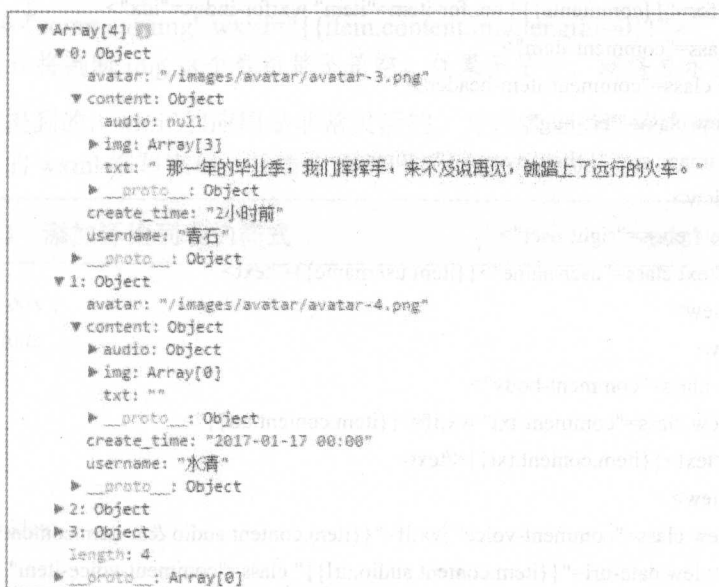


图 7-6 《从童年呼啸而过的火车》相关评论数据的打印结果

comments 数组中包含了 4 个 Object 对象，每一个对象代表着一条评论。

我们的评论支持文本、图片和录音 3 种类型。开发者可以对比一下图 7-6 中的 content 属性，在该属性下，img 数组是评论中的图片；txt 是评论中的文本；而 audio 是评论中的音频。注意，img 是数组，txt 是字符串，而 audio 是对象。

对于一条评论，有以下几条规则：

- 图片类型评论最多只能包含 3 张图片。
- 音频类型评论只能包含一条音频。
- 一条评论可以同时包含文字和图片。
- 音频类型评论不能包含文字和图片。

这样，我们就完成了获取并绑定文章评论数据这个步骤。在下一节中，我们将编写 post-comment 页面的 wxml 和 wxss 文件来显示这些已绑定的数据。

7.8 显示文章评论数据

读取到文章评论数据后，我们需要编写 post-comment 页面的 wxml 和 wxss 文件以显示这些数据。在 post-comment.wxml 中增加以下代码：

代码清单 7-22 显示评论数据的 wxml 代码

post.wxml

```
<view class="comment-detail-box">
  <view class="comment-main-box">
    <view class="comment-title">评论……（共{{comments.length}}条）</view>
    <block wx:for="{{comments}}" wx:for-item="item" wx:for-index="idx">
      <view class="comment-item">
        <view class="comment-item-header">
          <view class="left-img">
            <image src="{{item.avatar}}"></image>
          </view>
          <view class="right-user">
            <text class="user-name">{{item.username}}</text>
          </view>
        </view>
        <view class="comment-body">
          <view class="comment-txt" wx:if="{{item.content.txt}}">
            <text>{{item.content.txt}}</text>
          </view>
          <view class="comment-voice" wx:if="{{item.content.audio && item.content.audio.url}}">
            <view data-url="{{item.content.audio.url}}" class="comment-voice-item"
              catchtap="playAudio">
```



```

        <image src="/images/icon/wx_app_voice.png" class="voice-play"></image>
        <text>{{item.content.audio.timeLen}}</text>
    </view>
</view>
<view class="comment-img" wx:if="{{item.content.img.length!=0}}">
    <block wx:for="{{item.content.img}}" wx:for-item="img">
        <image src="{{img}}" mode="aspectFill"></image>
    </block>
</view>
</view>
<view class="comment-time">{{item.create_time}}</view>
</view>
</block>
</view>
</view>

```

整个代码里所用到的知识点都已经在前面的内容中讲到过，每条评论都会对 3 种类型的评论做处理。

我们重点来看看上述代码中的几个 wx:if 条件渲染：

- `<view class="comment-txt" wx:if="{{item.content.txt}}">`
这里的 wx:if 将判断 item.content.txt 有没有值，如果没有，那么整个 view 都不会显示；如果有，就将显示文字评论。
- `<view class="comment-voice" wx:if="{{item.content.audio && item.content.audio.url}}">`
这里的 wx:if 将判断 audio 是不是空值，如果不是空值接着判断 audio 这个对象的 url 有没有值，只有满足这两个条件才会显示音频评论。
- `<view class="comment-img" wx:if="{{item.content.img.length!=0}}">`
这里的 wx:if 将判断 img 这个数组是不是空，如果不是空，就将显示多张图片。

正如我们之前提到的，wx:if 的应用是非常灵活的，开发者应当理解这种用法。

post-comment 的 wxml 编写完后，接着来添加评论列表的样式。

代码清单 7-23 添加评论列表的样式

post-comment.wxss

```

.comment-detail-box {
    position: absolute;
    top: 0;
    left: 0;
    bottom: 0;
    right: 0;
    overflow-y: hidden;
}

.comment-main-box {

```

```
position: absolute;
top: 0;
left: 0;
bottom: 100rpx;
right: 0;
overflow-y: auto;
}

.comment-item {
margin: 20rpx 0 20rpx 24rpx;
padding: 24rpx 24rpx 24rpx 0;
border-bottom: 1rpx solid #f2e7e1;
}

.comment-item:last-child {
border-bottom: none;
}

.comment-title {
height: 60rpx;
line-height: 60rpx;
font-size: 28rpx;
color: #212121;
border-bottom: 1px solid #ccc;
margin-left: 24rpx;
padding: 8rpx 0;
font-family: Microsoft YaHei;
}

.comment-item-header {
display: flex;
flex-direction: row;
align-items: center;
}

.comment-item-header .left-img image {
height: 80rpx;
width: 80rpx;
}

.comment-item-header .right-user {
margin-left: 30rpx;
line-height: 80rpx;
```



```

}

.comment-item-header .right-user text {
  font-size: 26rpx;
  color: #212121;
}

```

```

.comment-body {
  font-size: 26rpx;
  line-height: 26rpx;
  color: #666;
  padding: 10rpx 0;
}

```

```

.comment-txt text {
  line-height: 50rpx;
}

```

```

.comment-img {
  margin: 10rpx 0;
}

```

```

.comment-img image {
  max-width: 32%;
  margin-right: 10rpx;
  height: 220rpx;
  width: 220rpx;
}

```

```

.comment-voice-item {
  display: flex;
  flex-direction: row;
  align-items: center;
  width: 200rpx;
  height: 64rpx;
  border: 1px solid #ccc;
  background-color: #fff;
  border-radius: 6rpx;
}

```

```

.comment-voice-item .voice-play {

```

```

  height: 64rpx;
  width: 64rpx;

```

```
}  
  
.comment-voice-item text {  
  margin-left: 60rpx;  
  color: #212121;  
  font-size: 22rpx;  
}  
  
.comment-time {  
  margin-top: 10rpx;  
  color: #ccc;  
  font-size: 24rpx;  
}
```

以上代码中有部分样式使用了 `position:absolute`，这是为了后面编写新增评论的功能而准备的。

保存运行代码，`post-comment` 将显示如图 7-7 所示的界面。



图 7-7 评论列表的显示效果

如果此时我们尝试去点击第二条评论的语言播放，就会发现它并没有效果，原因是初始化数据中的语音给的是一个假的 `url`。这里只是为了展现语音评论的显示效果，在后面的章节中我们将真实地新增和播放语音评论。

7.9 实现图片预览

在 7.8 节中，我们所有的图片都以固定尺寸显示，并将 image 的 mode 设置为了 aspectFill。本节我们将为图片添加预览功能。

无须自己编写图片预览插件，小程序已经为我们提供好了图片预览的接口：wx.previewImage(object)。它有以下两个重要属性：

- current 当前显示图片的链接，不填则默认为 urls 的第一张。
- urls 需要预览的图片链接列表，类型为数组。

这里要注意的是，urls 是一个数组，可以支持多张图片。它实际上类似于一个相册，可以左右滑动查看多张图片。

修改 post-comment.wxml 中 class="comment-img" 这个 view 组件内容。

代码清单 7-24 实现图片预览

post-comment.wxml

```
<view class="comment-img" wx:if="{{item.content.img.length!=0}}">
  <block wx:for="{{item.content.img}}" wx:for-item="img" wx:for-index="imgIdx">
    <image src="{{img}}" mode="aspectFill" catchtap="previewImg" data-comment-idx="{{idx}}"
data-img-idx="{{imgIdx}}"></image>
  </block>
</view>
```

以上代码在原有代码的基础上增加了以下属性：

- 在每一张 image 图片上注册一个事件 catchtap="previewImg"，用来相应点击图片的操作。
- 在 block 标签上新增 wx:for-index="imgIdx"，用以定义图片序号。
- 在每一张 image 图片上绑定了一个自定义属性：data-comment-idx="{{idx}}"，用来绑定当前评论在评论数组中的序号，并在 previewImg 方法中获取这个序号。idx 已在 <block wx:for="{{comments}}" wx:for-item="item" wx:for-index="idx"> 标签中定义。
- 在每一张 image 图片上绑定一个自定义属性 data-img-idx="{{imgIdx}}"，用来绑定图片在图片数组中的序号，并在 previewImg 方法中获取这个序号。

接着在 post-comment.js 中实现 previewImg 这个方法。

代码清单 7-25 定义 previewImg 方法

post-comment.js

```
//预览图片
previewImg: function (event) {
  //获取评论序号
  var commentIdx = event.currentTarget.dataset.commentIdx,
  //获取图片在图片数组中的序号
```

```

imgIdx = event.currentTarget.dataset.imgIdx,
//获取评论的全部图片
imgs = this.data.comments[commentIdx].content.img;
wx.previewImage({
  current: imgs[imgIdx], // 当前显示图片的 http 链接
  urls: imgs // 需要预览的图片 http 链接列表
})
}

```

注意 `wx.previewImage` 的用法，它接收一个 `object` 对象，对象的 `urls` 数组定义了一组需要预览的图片 `url`；而 `current` 定义了当前展示的图片 `url`。

完成以上代码后，保存刷新项目。

这时，我们会发现点击评论中的某一张图片后会打开图片预览窗口，但图片并不会显示出来。

`wx.previewImage` 在当前 130400 版本中有以下几个情况会造成无法预览图片：

- `wx.previewImage` 只能预览位于网络中的图片，而无法预览本地图片。我们初始化数据中的图片是位于本地的，所以无法预览。开发者可以将 `data.js` 文件中的文章评论图片地址更换为以 `http` 开头的网络地址。
- 除了网络地址和本地地址，还有一种地址是小程序的临时文件地址，对于这样的临时文件地址，同样在模拟器中无法预览，但在真机中却可以预览。关于临时文件地址，我们将在后面学习 `wx.chooseImage` 方法时看到。

注意，在目前的 130400 版本中，本地文件既无法在模拟器中预览，也无法在真机中预览；临时地址文件无法在模拟器中预览，却可以在真机中预览。

130400 版本的下一个版本已经放出了 `beta` 测试版，官方文档在版本说明中指出有可能会修复 `wx.previewImage` 不支持 `localId` 的问题。经下载测试版测试，开发工具中可以预览来自临时文件地址的图片，但依然不可以预览本地图片。

总体来说，这些怪异的现象对我们开发小程序影响不大，因为在真实项目中评论显示的图片都来自于服务器，所以无论是在模拟器中还是在真机中都可以预览。

7.10 实现提交评论的界面

在前几节中我们完成了评论的显示功能。在这一节中我们将实现如何提交一条文本类型的评论。提交评论的功能区域如图 7-8 所示。

点击提交功能区域最左侧的声音图标将可以由发送文本切换到发送语音，点击右边的加号图标将可以选择图片和拍照。



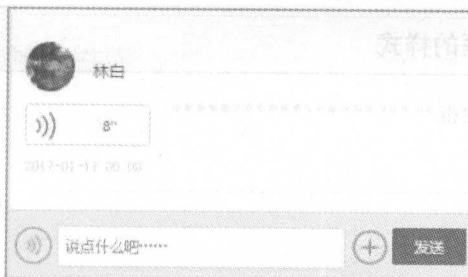


图 7-8 提交功能区域示意图

我们一步步地完成以上各项功能。

首先需要完成的是文本类型评论的提交功能。在 `post-comment.wxml` 文件中新增一段代码，以显示评论区域。

代码清单 7-26 评论框的骨架

post-comment.wxml

```
<view class="comment-detail-box">
  <view class="comment-main-box">
    <!--省略若干代码-->
  </view>

  <view class="input-box">
    <view class="send-msg-box">
      <view hidden="{{useKeyboardFlag}}" class="input-item">
        <image src="/images/icon/wx_app_keyboard.png" class="comment-icon keyboard-icon"
catchtap="switchInputType"></image>
        <input class="input speak-input {{recodingClass}}" value="按住 说话" disabled="disabled"
catchtouchstart="recordStart" catchtouchend="recordEnd" />
      </view>
      <view hidden="{{!useKeyboardFlag}}" class="input-item">
        <image class="comment-icon speak-icon" src="/images/icon/wx_app_speak.png"
catchtap="switchInputType"></image>
        <input class="input keyboard-input" value="{{keyboardInputValue}}"
bindinput="bindCommentInput" placeholder="说点什么吧....." />
      </view>
      <image class="comment-icon add-icon" src="/images/icon/wx_app_add.png"
catchtap="sendMoreMsg"></image>
      <view class="submit-btn" catchtap="submitComment">发送</view>
    </view>
  </view>
</view>
```

接着编写 `post-comment` 页面的样式，在 `post-comment.wxss` 文件中新增以下样式：

代码清单 7-27 评论框的样式

post-comment.wxss

/*****评论框*****/

```
.input-box{
  position: absolute;
  bottom: 0;
  left: 0;
  right: 0;
  background-color: #EAE8E8;
  border-top: 1rpx solid #D5D5D5;
  min-height: 100rpx;
  z-index: 1000;
}

.input-box .send-msg-box{
  width: 100%;
  height: 100%;
  display: flex;
  padding: 20rpx 0;
}

.input-box .send-more-box{
  margin: 20rpx 35rpx 35rpx 35rpx;
}

.input-box .input-item{
  margin: 0 5rpx;
  flex: 1;
  width: 0%;
  position: relative;
}

.input-box .input-item .comment-icon{
  position: absolute;
  left: 5rpx;
  top: 6rpx;
}

.input-box .input-item .input{
  border: 1rpx solid #D5D5D5;
  background-color: #fff;
  border-radius: 3px;
  line-height: 65rpx;
  margin: 5rpx 0 5rpx 75rpx ;
  font-size: 24rpx;
  color: #838383;
  padding: 0 2%;
```



```
}  
.input-box .input-item .keyboard-input{  
    width: auto;  
    max-height: 500rpx;  
    height: 65rpx;  
    word-break:break-all;  
    overflow:auto;  
}  
.input-box .input-item .speak-input{  
    text-align: center;  
    color: #212121;  
    height: 65rpx;  
}  
  
.input-box .input-item .recoding{  
    background-color: #ccc;  
}  
  
.input-box .input-item .comment-icon.speak-icon{  
    height: 62rpx;  
    width: 62rpx;  
}  
.input-box .input-item .comment-icon.keyboard-icon{  
    height: 60rpx;  
    width: 60rpx;  
    left:6rpx;  
}  
  
.input-box .add-icon{  
    margin:0 5rpx;  
    height: 65rpx;  
    width: 65rpx;  
    transform: scale(0.9);  
    margin-top: 2px;  
}  
.input-box .submit-btn{  
    font-size: 24rpx;  
    margin-top: 5rpx;  
    margin-right: 8rpx;  
    line-height: 60rpx;  
    width: 120rpx;  
    height: 60rpx;  
    background-color: #4A6141;  
    border-radius:5rpx;
```



```
color: #fff;
text-align: center;
font-family: Microsoft Yahei;
}

.send-more-box .more-btn-item{
display: inline-block;
width: 110rpx;
height: 145rpx;
margin-right: 35rpx;
text-align: center;
}

.more-btn-main{
width: 100%;
height: 60rpx;
text-align: center;
border: 1px solid #D5D5D5;
border-radius: 10rpx;
background-color: #fbfbfc;
margin: 0 auto;
padding: 25rpx 0
}

.more-btn-main image{
width: 60rpx;
height: 60rpx;
}

.send-more-box .more-btn-item .btn-txt{
color: #888888;
font-size: 24rpx;
margin: 10rpx 0;
}

.send-more-result-main{
margin-top: 30rpx;
}

.send-more-result-main .file-box{
margin-right: 14rpx;
height: 160rpx;
width: 160rpx;
position: relative;
display: inline-block;
```



```

}

.send-more-result-main .file-box.deleting{
  animation:deleting 0.5s ease;
  animation-fill-mode: forwards;
}

@keyframes deleting {
  0%{
    transform: scale(1);
  }
  100%{
    transform: scale(0);
  }
}

.send-more-result-main image{
  height: 100%;
  width: 100%;
}

.send-more-result-main .remove-icon{
  position: absolute;
  right: 5rpx;
  top: 5rpx;
}

.send-more-result-main .file-box .img-box {
  height: 100%;
  width: 100%;
}

```

保存后，可以看到评论框的大致模样。当然，我们还没有为评论框编写任何逻辑代码，此时开发工具有可能会报错。

需要注意的是，代码清单 7-27 中的部分样式在代码清单 7-26 中并未全部用到，但我们在后续的代码中将会使用到。

下面我们来对 wxml 里的新增代码关键部分做一些解释。

`<view hidden="{{useKeyboardFlag}}" class="input-item">`表示录音输入框。

`<view hidden="{{!useKeyboardFlag}}" class="input-item">`表示键盘输入框。

以上两个评论框由 `useKeyboardFlag` 这个 Boolean 变量来控制显示或者隐藏。`useKeyboardFlag` 变量将由 `catchtap="switchInputType"`这个事件来控制。

`catchtouchstart="recordStart"`和 `catchtouchend="recordEnd"`将开启录音和结束录音。

`<input class="input keyboard-input">` 实现文字内容的录入。关于 `input` 组件的使用，我们将在后面详细介绍。

catchtap="sendMoreMsg"将实现向内容中添加图片和拍照选择框的功能。

catchtap="submitComment"将实现评论内容的最终发送功能。

7.11 wx:if 与 hidden 控制元素显示和隐藏

在小程序中，最常用的显示/隐藏 UI 元素的方法有两种：一种是之前我们介绍的 wx:if，另外一种 hidden。我们特别在代码清单 7-26 中使用了 hidden 这种方式来控制元素的显示和隐藏效果。

在代码清单 7-26 中，注意以下两段代码。

- `<view hidden="{{useKeyboardFlag}}" class="input-item">`
- `<view hidden="{{!useKeyboardFlag}}" class="input-item">`

hidden 的使用方式与 wx:if 类似，都是通过一个状态变量来控制元素的显示和隐藏。

当 useKeyboardFlag 为 true 时，第 1 个 view(hidden="{{useKeyboardFlag}}") 将被隐藏，而第 2 个 view(hidden="{{!useKeyboardFlag}}") 将被显示。当 useKeyboardFlag 为 false 时，则第 1 个 view 被显示，第 2 个 view 被隐藏。

那么 wx:if 和 hidden 之间有什么异同吗？

wx:if 的切换和渲染机制较为复杂。当 wx:if 进行切换时，MINA 框架有一个局部渲染的过程，它会确保条件块在切换时销毁或重新渲染。

同时 wx:if 也是惰性的，如果初始渲染条件为 false，那么框架什么也不做，在条件第一次变成真的时候才开始局部渲染。

相比之下，hidden 就简单得多了，组件始终会被渲染，只是简单地控制显示与隐藏。

一般来说，wx:if 有更高的切换消耗，而 hidden 有更高的初始渲染消耗。因此，在需要频繁切换的情景下用 hidden 更好，在运行时条件不大可能改变时用 wx:if 较好。

7.12 实现文字评论框和语音评论框的切换

编写完页面的 wxml 和 wxss 文件后，我们来继续编写这些组件的业务逻辑。

首先实现“按住说话”和“说点什么吧...”这两个组件的切换效果。之前我们提到过，实现语音和文字评论框切换的效果关键是控制 useKeyboardFlag 这个变量。

代码清单 7-28 初始化 useKeyboardFlag

post-comment.js

```
data: {
  useKeyboardFlag: true,
},
```



在 `post-comment.js` 文件的 `Page(object)` 方法的 `data` 属性下新增一个 `useKeyboardFlag` 变量作为初始化的数据绑定变量。`useKeyboardFlag` 初始值为 `true` 将导致评论框默认显示为键盘类型的输入框。

接着编写 `switchInputType` 方法来切换 `useKeyboardFlag` 这个控制变量。

代码清单 7-29 切换 `useKeyboardFlag`

`post-comment.js`

```
//切换语音和键盘输入
switchInputType: function (event) {
    this.setData({
        useKeyboardFlag: !this.data.useKeyboardFlag
    })
}
```

此时，点击评论框最左侧的小图标将可以实现语音评论框和文字评论框的相互切换效果。

接着实现发送文字评论的功能。在实现发送文字评论功能之前，我们需要学习一个非常重要的组件：`input` 组件。

7.13 input 组件

`input` 组件是最为重要的数据输入组件，比如我们这里要输入用户的评论信息时就需要用到这个组件。`input` 组件目前拥有以下若干属性：

- `value` `String` 类型，设置 `input` 输入框的初始内容。
- `type` `String` 类型，`input` 组件目前有 4 种类型，即 `text`、`number`、`idcard`、`digit`，默认是 `text` 类型。
- `password` `Boolean` 类型，如果设置为 `true`，就会用 `*` 号来遮蔽输入，默认为 `false`。
- `placeholder` `String` 类型，输入框为空时的占位符。所谓占位符，就是当输入框内没有任何用户输入时默认显示的文字，比如 `post-comment` 页面文字输入框中默认显示的“说点什么吧...”。
- `placeholder-style` `String` 类型，指定 `placeholder` 的样式。可以将 `placeholder` 的样式编写在这里，形如组件中的 `style` 属性一样，只不过 `placeholder-style` 属性专门用来编写 `placeholder` 样式，而非编写整个 `input` 的样式。比如你可以这样写：`placeholder-style="color:#333;font-size:24rpx"`。
- `placeholder-class` `String` 类型，如果你不想在组件的标签上写样式，就可以使用这个属性来指定一个已编写好的 `CSS` 类名，如同标签的 `class` 属性一样使用。
- `disabled` `Boolean` 类型，用于设置是否禁用 `input` 组件，默认值为 `false`。
- `maxlength` `Number` 类型，最大输入长度。设置为 `-1` 的时候不限制最大长度，默认值为 `140`。

- `cursor-spacing` Number 类型, 指定光标与键盘的距离, 单位是 px。取 `input` 距离底部的距离和 `cursor-spacing` 指定的距离的最小值作为光标与键盘的距离, 默认值为 0。
- `focus` Boolean 类型, 自动获取焦点并拉起键盘, 默认值为 `false`。如果某个页面的 `input` 组件包含这个 `focus` 属性, 且将其值设置为 `true`, 那么当进入这个页面后光标将自动定位到 `input` 中, 且会自动拉起输入键盘。

以上是 `input` 的所有属性。接下来我们看看 `input` 组件的 4 个事件, 这 4 个事件才是 `input` 组件的重点和难点:

- `bindinput`
- `bindfocus`
- `bindblur`
- `bindconfirm`

注意以上事件和我们常用的 `catch`、`bind` 开头的通用事件是有区别的。它们是由 MINA 框架直接指定的, 不需要在事件名称前再添加 `catch` 和 `bind`, 千万不要写成 `bindbindinput` 或者 `catchbindinput`。

此外, 以上 4 个事件都属于非冒泡事件, 这是它们和 `catch`、`bind` 等通用前缀事件的重要区别。形如 `catchtap` 等事件通常都是冒泡事件。

首先来看看 `bindinput`。`bindinput` 事件较为特殊, 具有以下几个特点:

- 当用户输入字符时触发。
- 每当用户输入或者删除一个字符时, `bindinput` 事件都会触发一次。
- 可以在事件响应函数中使用 `return` 返回一个字符或者字符串, 该字符串将替换 `input` 输入框的显示文本。
- 它非常适合用来做“即时搜索”的功能。

`bindfocus`, 当 `input` 组件获取焦点时触发。我们将在本书后面的电影部分看到这个事件用法。

`bindblur`, 当 `input` 组件失去焦点时触发。

`bindconfirm`, 122100 版本新增事件, 专门用来响应真机上点击键盘“完成”按钮的事件。`input` 输入值都是在事件对应的响应函数中使用 `event.detail.value` 来获取的。

7.14 bindinput 事件

考虑到一些特殊的输入法键盘或者用户的习惯, 我们除了支持点击真机键盘上的“完成”按钮发送文字评论外, 还实现了一个小程序里的“发送”按钮。

我们先来实现自定义的“发送”按钮。实现自定义发送评论功能的第一步就是能够在 js 中获取 `input` 的 `value` 输入值。我们使用 `bindinput` 事件来获取 `input` 的输入值。

在 `post-comment.js` 文件中新增以下代码:



代码清单 7-30 响应 bindinput 事件

post-comment.js

```
//获取用户输入
bindCommentInput: function (event) {
    var val = event.detail.value;
    console.log(val);
    this.data.keyboardInputValue = val;
},
```

使用事件的 event 对象下的 detail.value 来获取 input 的输入值，并将这个值保存在 this.data 中。

除此之外，我们在代码中加入了一段 `console.log(val)`，一起来看看 `bindinput` 事件是如何响应用户输入的。

在 input 输入框中不断输入任意字母, 比如'q', 再不断地删除'q'。在 Console 面板中将看到如图 7-9 所示的输出。

可以看到，每次输入一个 `q` 都会触发 `bindinput` 事件，并 `console` 出当前的 `input` 值；每次删除一个字符 `q` 同样会触发 `bindinput` 事件，并输出当前的 `input` 值。

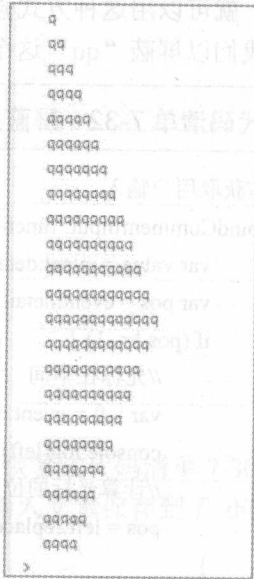


图 7-9 bindinput 事件的响应机制

7.15 屏蔽评论关键字

`bindinput` 还有一个有意思的特性,就是在事件响应函数中可以 `return` 一个值来代替当前的输入值,并显示在 `input` 中。下面一起来看一下效果。

代码清单 7-31 bindinput 的 return 值

post-comment.js

```
bindCommentInput: function (event) {
    var val = event.detail.value;
    return val + "+"
}
```

将 `bindCommentInput` 函数内部的代码临时更改为以上代码（请注意在测试完毕后还原成之前的代码）。Data comment.txt

保存代码后，在 input 组件中不断地输入字符“q”，input 组件将显示如图 7-10 所示的内容。

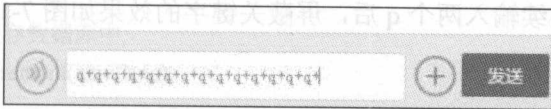


图 7-10 bindinput 事件的 return 值



很明显，每次输入 q 后都会被更改为 q+。

这个 return 的机制非常适合用来过滤关键字。如果不想让用户输入“qq”“微信号”等关键字，就可以用这种方式强制过滤掉。

我们以屏蔽“qq”这个关键字为例来看看如何实现屏蔽关键字。

代码清单 7-32 屏蔽关键字“qq”

post-comment.js

```
//获取用户输入
bindCommentInput: function (event) {
  var value = event.detail.value;
  var pos = event.detail.cursor;
  if (pos !== -1) {
    //光标在中间
    var left = event.detail.value.slice(0, pos);
    console.log(left);
    //计算光标的位置
    pos = left.replace(/qq/g, '*').length;
  }

  //直接返回对象，可以对输入进行过滤处理，同时可以控制光标的位置
  return {
    value: value.replace(/qq/g, '*'),
    cursor: pos
  }
}
```

以上代码实现了当用户输入“qq”时，自动被替换成“*”。注意，最后 return 的是一个 object 对象，该对象的 value 表示要替换的文本值，而 cursor 表示光标所处的位置。

其实我们不需要关心 cursor 光标的位置，以上代码只是为了告诉开发者可以控制光标的位置。如果只想屏蔽掉关键字，只需要以下几行代码：

代码清单 7-33 简化屏蔽掉“qq”关键字的代码

post.comment.js

```
//获取用户输入
bindCommentInput: function (event) {
  var value = event.detail.value;
  return value.replace(/qq/g, '*')
}
```

当用户在 input 中连续输入两个 q 后，屏蔽关键字的效果如图 7-11 所示。



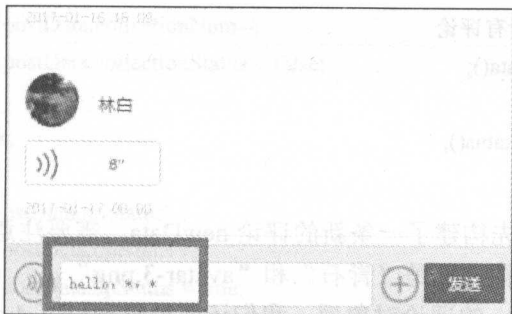


图 7-11 屏蔽“qq”关键字

7.16 实现自定义发送按钮

在实现自定义发送按钮功能之前，请将 `bindCommentInput` 函数恢复成代码清单 7-30 中的代码。在代码清单 7-30 中，我们成功地获取了用户的输入，并将输入文本保存到了 `this.data` 变量中。

发送按钮的事件响应函数是 `submitComment`。在 `post-comment.js` 的 `Page` 函数中添加以下代码：

代码清单 7-34 实现 `submitComment` 方法

post-comment.js

```
// 提交用户评论
submitComment: function (event) {
  var newData = {
    username: "青石",
    avatar: "/images/avatar/avatar-3.png",
    // 评论时间
    create_time: new Date().getTime() / 1000,
    // 评论内容
    content: {
      txt: this.data.keyboardInputValue
    },
  };
  if (!newData.content.txt) {
    // 如果没有评论内容，就不执行任何操作
    return;
  }
  //保存新评论到缓存数据库中
  this.dbPost.newComment(newData);
  //显示操作结果
  this.showCommitSuccessToast();
}
```

```

//重新渲染并绑定所有评论
this.bindCommentData();
//恢复初始状态
this.resetAllDefaultStatus();
}

```

submitComment 中首先构建了一条新的评论 newData。需要注意的是，newData 中硬编码了当前用户的用户名，头像分别是“青石”和“avatar-3.png”。

当组装完这个 newData 的评论对象后，我们还需要经过以下 4 个步骤：

- 步骤 01 将 newData 保存到缓存数据库，以便下次打开评论页面时可以显示这条 newData。
- 步骤 02 显示评论发表成功的提示。
- 步骤 03 将当前发表的评论添加到评论列表中，并显示这条新添加的评论。
- 步骤 04 清空 input 组件，准备接收下一条评论。

以上 4 个步骤分别对应着代码清单 7-34 中最末尾的 4 个方法调用。

首先来完成步骤 1。在 DBPost 中新增 newComment 方法，用来保存新评论到缓存数据库中。

代码清单 7-35 编写 newComment 方法

DBPost.js

```

/*发表评论*/
newComment(newComment) {
  this.updatePostData('comment', newComment);
}

```

该方法内部再一次调用了 this.updatePostData 方法。我们来修改一下 updatePostData 方法，让其能够支持“新增评论”。完整的 updatePostData 如下：

代码清单 7-36 在 updatePostData 中处理新评论

DBPost.js

```

//更新本地的点赞、评论信息、收藏、阅读量
updatePostData(category, newComment) {
  var itemData = this.getItemById(),
      postData = itemData.data,
      allPostData = this.getAllPostData();
  switch (category) {
    case 'collect':
      //处理收藏
      if (!postData.collectionStatus) {
        //如果当前状态是未收藏
        postData.collectionNum++;
        postData.collectionStatus = true;
      } else {
        // 如果当前状态是收藏

```




```

        postData.collectionNum--;
        postData.collectionStatus = false;
    }
    break;
case 'up':
    if (!postData.upStatus) {
        postData.upNum++;
        postData.upStatus = true;
    } else {
        postData.upNum--;
        postData.upStatus = false;
    }
    break;
case 'comment':
    postData.comments.push(newComment);
    postData.commentNum++;
    break;
default:
    break;
}
allPostData[itemData.index] = postData;
this.execSetStorageSync(allPostData);
return postData;
}

```

相比之前的 `updatePostData` 方法，我们为该方法新增了一个参数 `newComment`，用以接收新的评论数据；接着在 `case` 中新增了一个 `case 'comment'` 分支，用来处理新增评论。

这样步骤 1 就完成了。接着我们来编写步骤 2：显示评论发表成功的提示。

在 `post-comment.js` 中新增 `showCommitSuccessToast` 方法。

代码清单 7-37 编写新增评论成功的提示方法

post-comment.js

```

//评论成功
showCommitSuccessToast: function () {
    //显示操作结果
    wx.showToast({
        title: "评论成功",
        duration: 1000,
        icon: "success"
    })
}

```

以上代码将完成步骤 2，继续来编写步骤 3：将当前发表的评论添加到评论列表中。



在拥有 DOM 节点的对象中，比如在传统网页中，如果我们想插入一条评论，需要新增一个 DOM 节点，并将这个 DOM 节点 insert 插入到 DOM 数组中。

在小程序中，我们没有 DOM，也只有一种方式可以操作数据，即数据绑定。不存在“新增一个 DOM，再将 DOM 节点插入到 DOM 对象数组中”这样的思路。

如果我们需要在已有的 n 条评论中插入一条评论，并在 UI 中显示这 $n+1$ 条评论，我们只能将这 $n+1$ 条评论全部重新做数据绑定。我们在实际项目中会有大量插入元素的需求，请开发者牢记这个思路。

在 post-comment.js 的 Page 方法中新增以下方法：

代码清单 7-38 重新绑定评论数据

post-comment.js

```
bindCommentData: function () {
    var comments = this.dbPost.getCommentData();
    // 绑定评论数据
    this.setData({
        comments: comments
    });
}
```

以上方法重新去缓存数据库中加载全部的评论并再次使用 this.setData 将全部评论进行数据绑定。

最后完成步骤 4：清空 input 组件，准备接收下一条评论。

清空 input 组件的方法很简单，将 input 的 value 属性重置为空字符串即可。在 post-comment.js 的 Page 方法中添加以下代码：

代码清单 7-39 重置 input 组件的输入值

post-comment.js

```
//将所有相关的按钮状态、输入状态都恢复到初始状态
resetAllDefaultStatus: function () {
    //清空评论框
    this.setData({
        keyboardInputValue: ""
    });
}
```

resetAllDefaultStatus 方法重新绑定了 keyboardInputValue，将其值设置为空字符串。注意，keyboardInputValue 在之前的代码中已经被绑定到了 input 组件的 value 属性上了。

完成以上 4 个步骤后保存并运行代码。先在输入框中输入一段文字，再点击右侧的发送按钮，一条评论就会出现在评论列表中，且这条评论位于评论列表的顶部。



7.17 同时支持模拟器回车、真机点击“完成”发送评论

到目前为止，我们已经实现了自定义发送按钮发送评论的功能。我们再来实现在模拟器中以敲击键盘回车发送评论和在真机中点击键盘“完成”发送评论的功能。

如果想在模拟器中实现回车发送评论消息的功能，可以使用以下几个 input 事件：

- bindchange
- bindblur
- bindconfirm

bindchange 在早期版本中是有的，但官方在后来的文档中去掉了这个事件，截至 130400 版本，bindchange 依然有效。虽然它依然有效，但是不建议使用官方文档未明确说明的事件。

bindblur 可以触发回车的原理是，点击回车后，input 组件将失去焦点，从而触发 bindblur 事件。

bindconfirm 可以在真机上响应键盘的“完成”点击事件，同时也可以可以在模拟器中响应键盘的“回车”敲击事件。

拿 bindconfirm 举例，修改 input 组件为如下代码：

代码清单 7-40 添加 bindconfirm 事件

post-comment.wxml

```
<input class="input keyboard-input" value="{{keyboardInputValue}}"
  bindconfirm="submitComment"
  bindinput="bindCommentInput"
  placeholder="说点什么吧……" />
```

我们仅仅在 input 组件上新增了一个 bindconfirm 事件，这个事件的响应函数与自定义发送按钮所响应的事件函数相同，同样都是 submitComment。这样就可以同时实现自定义发送按钮发送评论、模拟器回车发送评论和真机上点击“完成”发送评论的功能。

开发者还可以依次将 bindconfirm 更换为 bindblur 和 bindchange，看一下它们的效果。虽然以上 3 个事件都可以实现回车的效果，但建议使用 bindconfirm 事件作为 input 的“确定”事件。

7.18 图片与拍照评论的界面实现

我们接着实现图片和拍照评论。先来实现以下效果：当点击 + 号按钮后，出现选择图片和拍照的界面，如图 7-12 所示。

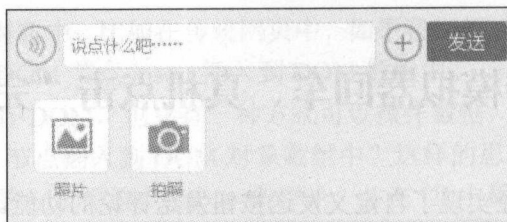


图 7-12 出现照片和拍照界面

首先在 post-comment.wxml 中添加图 7-12 所示的骨架代码。

代码清单 7-41 选择图片与拍照面板 wxml 代码

post-comment.wxml

```
<view class="send-more-box" hidden="{{!sendMoreMsgFlag}}">
  <!--选择图片和拍照的按钮-->
  <view class="send-more-btms-main">
    <view class="more-btn-item" catchtap="chooseImage" data-category="album">
      <view class="more-btn-main">
        <image src="/images/icon/wx_app_upload_image.png"/>
      </view>
      <text>照片</text>
    </view>
    <view class="more-btn-item" catchtap="chooseImage" data-category="camera">
      <view class="more-btn-main">
        <image src="/images/icon/wx_app_camera.png"/>
      </view>
      <text>拍照</text>
    </view>
  </view>

  <!--显示选择的图片-->
  <view class="send-more-result-main" hidden="{{chooseFiles.length==0}}">
    <block wx:for="{{chooseFiles}}" wx:for-index="idx">
      <!--如果删除其中一个，就对其添加 deleting 样式-->
      <view class="file-box {{deleteIndex==idx?'deleting:'}}">
        <view class="img-box">
          <image src="{{item}}" mode="aspectFill"/>
          <icon class="remove-icon" type="cancel" size="23" color="#B2B2B2" catchtap="deleteImage"
            data-idx="{{idx}}"/>
        </view>
      </view>
    </block>
  </view>
</view>
```



注意，以上代码位于<view class="input-box">这个 view 的内部，与<view class="send-msg-box">同级。如果无法找到这段代码的准确位置，请下载源代码查看。

对应的 CSS 代码已在代码清单 7-27 中添加完毕，不需要再次编写。

这段代码的开头部分有一段 hidden="{{!sendMoreMsgFlag}}"。sendMoreMsgFlag 变量将控制整个面板的显示和隐藏。默认状态下它是隐藏的，所以我们首先在 post-comment.js 的 Page 方法 data 属性下设置 sendMoreMsgFlag 的初始状态。

代码清单 7-42 设置 sendMoreMsgFlag 的初始状态

post-comment.js

```
data: {
  // 控制使用键盘还是发送语音
  useKeyboardFlag: true,
  // 控制 input 组件的初始值
  keyboardInputValue: "",
  // 控制是否显示图片选择面板
  sendMoreMsgFlag: false
}
```

sendMoreMsgFlag 的状态被我们设置成了 false。这样在默认状态下图片与拍照面板是不会显示的。

事实上，即使我们不在 data 下面设置 sendMoreMsgFlag，面板依然不显示，因为如果找不到这个变量，默认这个变量的取值就是 false。为了代码的可读性，建议开发者将变量明确地在 data 属性中标注出来。这样的习惯应该成为我们开发小程序的一个小小约定。

keyboardInputValue 变量也是基于这个原因而设置了一个空的字符串。keyboardInputValue 是之前控制 input 组件初始值的变量，我们在代码清单 7-39 中使用了这个变量，开发者可自行回顾一下。

接着实现 sendMoreMsg 方法，它将切换 sendMoreMsgFlag 变量，以实现面板的切换和隐藏。sendMoreMsg 方法已经被我们在代码清单 7-26 中注册到了以下图片的 catchtap 事件上。

```
<image class="comment-icon add-icon" src="/images/icon/wx_app_add.png" catchtap="sendMoreMsg">
</image>
```

在 post-comment.js 的 Page 中新增 sendMoreMsg 方法。

代码清单 7-43 编写 sendMoreMsg 方法

post-comment.js

```
//显示选择照片、拍照等按钮
sendMoreMsg: function () {
  this.setData({
    sendMoreMsgFlag: !this.data.sendMoreMsgFlag
  })
}
```

完成以上代码后，我们再次点击 + 号图标，拍照面板将动态地显示和隐藏。

7.19 实现从相册选择照片与拍照

在成功显示照片面板后，我们需要实现从相册选择照片和拍照上传照片的功能。小程序提供了一个 API: `wx.chooseImage(object)` 来实现这个功能。

我们首先在 `data` 变量中新增一个数组来保存已选择图片的 URL。

代码清单 7-44 新增 `chooseFiles` 变量

post-comment.js

```
data: {
  // 控制使用键盘还是发送语音
  useKeyboardFlag: true,
  // 控制 input 组件的初始值
  keyboardInputValue: "",
  // 控制是否显示图片选择面板
  sendMoreMsgFlag: false,
  // 保存已选择的图片
  chooseFiles: []
}
```

每选择一组图片，我们都会将图片的 URL 保存在 `chooseFiles` 这个数组中。

在代码清单 7-41 中，我们分别在“照片”和“拍照”这两个图片按钮上注册了同一个事件: `chooseImage` 事件。点击这两个图片按钮后将执行 `chooseImage` 方法。在 `post-comment.js` 中编写这个事件方法。

代码清单 7-45 实现选择图片与拍照功能

post-comment.js

```
//选择本地照片与拍照
chooseImage: function (event) {
  // 已选择图片数组
  var imgArr = this.data.chooseFiles;
  //只能上传 3 张照片，包括拍照
  var leftCount = 3 - imgArr.length;
  if (leftCount <= 0) {
    return;
  }
  var sourceType = [event.currentTarget.dataset.category],
    that = this;
  wx.chooseImage({
    count: leftCount,
    sourceType: sourceType,
    success: function (res) {
```



```

// 可以分次选择图片，但总数不能超过 3 张
that.setData({
  chooseFiles: imgArr.concat(res.tempFilePaths)
});
}
})
},

```

解释一下上述代码。我们选择一次评论内容中最多只允许发送 3 张照片。Leftcount 检测当前已经选择了多少图片，如果超过了 3 张就直接 return，不能再执行下面的 wx.chooseImage 方法。

如果还不到 3 张，就多次选择照片，一直到选满 3 张为止。

我们重点看下 wx.chooseImage 方法。

wx.chooseImage(object)接收两个主要的参数：count 和 sourceType。

count 指定一次最多可以选择多少张图片。

sourceType 指定是拍照生成照片还是从手机相册选择照片。注意，它是一个数组，可以有以下几个取值：

- ['album']
- ['camera']
- ['album', 'camera']

第一个取值将直接打开相册，并可以选择照片。注意，无论是在开发工具中还是在真机上都可以支持多选。

第二个取值将直接打开相机并拍照生成照片。注意，在开发工具中，camera 取值不会打开相机拍照，只会打开相册。

第三个取值将打开如图 7-13 所示的图片界面。

['album','camera']可以让用户自行选择是拍照还是从手机相册选择。需要注意的是，在开发工具中这个取值不会出现这样的选择框。开发工具中只会直接打开相册，因为开发工具没有拍照功能。



图 7-13 ['album','camera']取值效果

我们分别选取第一个和第二个取值，对应面板中的“照片”和“拍照”功能。

var sourceType = [event.currentTarget.dataset.category]这段代码决定了当前选择的是哪一种方式。

当选择照片或者拍照成功后，将进入 wx.chooseImage 的 success 方法中，success 方法将接收一个框架传入的 res 参数。在 success 方法中加入 console.log(res)，res 的输出结果如图 7-14 所示。

重点是 tempFilePaths 这个数组，它装载了我们选择的 3 张图片（来自相册或者拍照）的 URL。注意，这个 URL 是图片的临时地址。

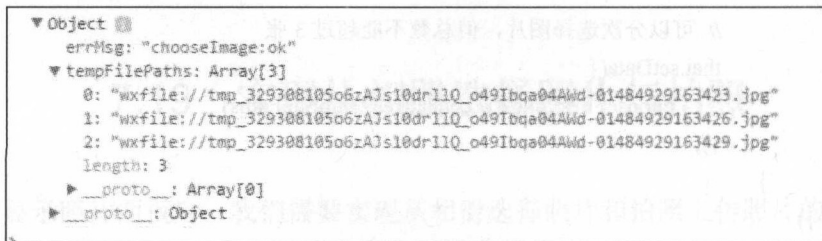


图 7-14 res 对象参数值

拿到图片的地址后，我们就可以将这些图片地址添加到 `imgArr` 中，并将 `imgArr` 绑定到 `chooseFiles` 变量中。一旦 `chooseFiles` 变量被绑定了数据，`wxml` 代码中的 `<block wx:for="{{chooseFiles}}" wx:for-index="idx">` 将循环显示这些图片，如图 7-15 所示。



图 7-15 选择图片的最终效果

需要注意的是，你并不需要一次选择 3 张，可以分多次选择，但最多只能选择 3 张。

7.20 icon 图片

在 7.19 节中，我们实现了图片的选择功能。注意看图 7-14，每张图片的右上角都有一张小图片，这个小图片并不是我们自定义的图片，而是由小程序的 `icon` 组件提供的图片。

在代码清单 7-41 中有一段代码：

```
<icon class="remove-icon" type="cancel" size="23"
color="#B2B2B2" catchtap="deleteImage" data-idx="{{idx}}" />
```

这段代码就是 `icon` 组件的用法。

`icon` 组件非常简单，只有 3 个属性：

- `type` `icon` 的类型，取值可以是 `success`、`success_no_circle`、`info`、`warn`、`waiting`、`cancel`、`download`、`search`、`clear`。我们在代码中选用的是 `cancel`。
- `size` `icon` 图片的大小，单位是 `px`。
- `color` 颜色，同 `CSS` 的 `color`。

图 7-16 所示的是不同类型的 icon 示意图。

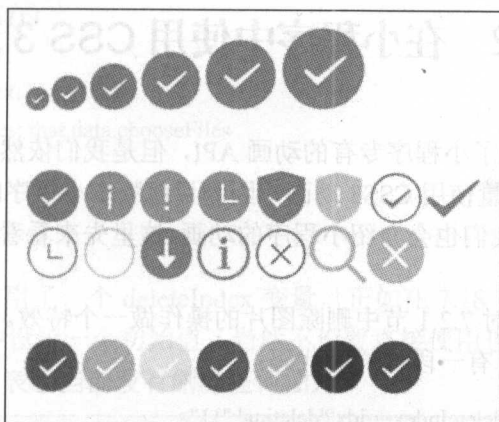


图 7-16 不同类型的 icon 示意图

如果开发者对项目样式要求不是很高，建议尽量选择 icon 组件。

7.21 删除已选择的图片

当我们选择好图片后，存在想删除图片并重新选择的可能，所以还需要支持图片的删除功能。

删除功能是通过点击图片右上角的 icon 图标来实现的。我们已经在 wxml 代码的 icon 组件中注册了 deleteImage 事件。

在 post-comment.js 中添加 deleteImage 方法。

代码清单 7-46 deleteImage 方法

post-comment.js

```
//删除已经选择的图片
deleteImage: function (event) {
  var index = event.currentTarget.dataset.idx,
      that = this;
  that.data.chooseFiles.splice(index, 1);
  that.setData({
    chooseFiles: that.data.chooseFiles
  });
}
```

删除图片的逻辑非常简单，只需要获取当前删除图片的序号，并将该图片的 URL 从 this.data.chooseFiles 数组中删除，重新绑定 chooseFiles 变量即可。

保存并重新运行代码，就可以通过点击已选择图片右上角的叉叉按钮删除图片了。删除图片后，可以重新选择图片。

7.22 在小程序中使用 CSS 3 动画

虽然 MINA 框架提供了小程序专有的动画 API，但是我们依然可以在小程序中使用 CSS3 动画。我们建议开发者尽量使用 CSS3 动画，因为目前版本小程序的动画存在较多 bug，使用起来并不是很方便。后面我们也会介绍小程序的动画，这里先来看看如何在小程序中使用 CSS3 动画。

我们使用 CSS3 动画对 7.2.1 节中删除图片的操作做一个特效。

在代码清单 7-41 中，有一段这样的代码：

```
<view class="file-box {{deleteIndex==idx?'deleting':''}}">
```

如果 deleteIndex 的值等于当前图片的序号，就说明该图片是要被删除的，需要添加一个 deleting 动画。

以下代码是 deleting 动画的 CSS 代码：

```
.send-more-result-main .file-box.deleting{
  animation:deleting 0.5s ease;
  animation-fill-mode: forwards;
}
```

```
@keyframes deleting {
  0%{
    transform: scale(1);
  }
  100%{
    transform: scale(0);
  }
}
```

这段动画已经在代码清单 7-27 中添加到了 post-comment.wxss 中，这里就无须再添加了。

我们需要做的就是删除图片时执行这段 CSS3 动画。修改 post-comment.js 中的 deleteImage 方法，以支持 CSS3 动画效果。

代码清单 7-47 修改 deleteImage 方法

post-comment.js

```
//删除已经选择的图片
deleteImage: function (event) {
  var index = event.currentTarget.dataset.idx,
      that = this;
  that.setData({
    deleteIndex: index
  });
}
```



```

that.data.chooseFiles.splice(index, 1);
setTimeout(function () {
  that.setData({
    deleteIndex: -1,
    chooseFiles: that.data.chooseFiles
  });
}, 500)
}

```

在新代码中，我们使用了一个 `deleteIndex` 变量，正如在 7.18 节中所讲的，最好将这些变量都在页面的 `data` 属性中设定一个初始值（当然不设置直接使用也可以）。在下面的代码中，将 `deleteIndex` 设置为 -1，表示当前没有删除任何图片。

代码清单 7-48 定义 `deleteIndex` 变量

post-comment.js

```

data: {
  // 控制使用键盘还是发送语音
  useKeyboardFlag: true,
  // 控制 input 组件的初始值
  keyboardInputValue: "",
  // 控制是否显示图片选择面板
  sendMoreMsgFlag: false,
  // 保存已选择的图片
  chooseFiles: [],
  // 被删除的图片序号
  deleteIndex: -1
}

```

在定义了 `deleteIndex` 变量后，在删除图片时首先使用 `this.setData` 方法更新 `deleteIndex` 变量值为当前删除图片的序号，使用 `this.setData` 将立即执行数据绑定，使被删除的图片立即添加并执行一个 `deleting` 动画（正如本节开头所分析的那样）。动画执行的时间为 500 毫秒，所以我们使用 `setTimeout` 函数延迟 500 毫秒后再真实地删除这张图片。

注意，一定要记得将 `deleteIndex` 的值恢复为 -1，否则动画执行将出错。

大多数动画的编写方式都如本节中所展示的方式那样，只是动画的效果不同而已。开发者可以举一反三，尝试不同的动画效果。

适度使用动画将大大提升客户端的体验，需要注意的是不要过度使用动画，特别是在一些性能不好的手机上，动画将大大消耗手机性能。建议能不用动画就尽量不用，毕竟小程序是即用即走，做太多动画的意义不大。



7.23 实现图片评论的发送

在前面我们已经完成了图片的选择和删除功能，接着实现发送图片评论。

实现发送图片评论的方式非常简单，只需要将当前 `this.data.chooseFiles` 所保存的图片地址存入数据缓存中，并重新渲染评论列表即可。我们来修改一下 `submitComment` 方法，这个方法我们已经在发送文字评论时实现过。

代码清单 7-49 修改 `submitComment` 方法

post-comment.js

```
submitComment: function (event) {
  var imgs = this.data.chooseFiles;
  var newData = {
    username: "青石",
    avatar: "/images/avatar/avatar-3.png",
    create_time: new Date().getTime() / 1000,
    content: {
      txt: this.data.keyboardInputValue,
      img: imgs
    },
  };
  if (!newData.content.txt && imgs.length === 0) {
    return;
  }
  //保存新评论到缓存数据库中
  this.dbPost.newComment(newData);

  //显示操作结果
  this.showCommitSuccessToast();
  //重新渲染并绑定所有评论
  this.bindCommentData();
  //恢复初始状态
  this.resetAllDefaultStatus();
}
```

相比之前的 `submitComment` 方法，我们只是将 `this.data.chooseFiles` 这个保存图片 URL 的数组加入到了 `newData` 中，这样 `newData` 中不仅仅包含有 `txt` 文本，还包括 `img` 图片。

在发送完图片评论后，还需要清空已选择的图片，并在此隐藏图片选择面板。修改 `resetAllDefaultStatus` 方法，代码如下：



代码清单 7-50 发送后重置组件状态

post-comment.js

//将所有相关的按钮状态、输入状态都恢复到初始化状态

```
resetAllDefaultStatus: function () {
```

```
  //清空评论框
```

```
  this.setData({
```

```
    keyboardInputValue: "",
```

```
    chooseFiles: [],
```

```
    sendMoreMsgFlag: false
```

```
  });
```

```
}
```

相比之前只重置了 `keyboardInputValue` 变量，这里还重置了 `chooseFiles` 变量和 `sendMoreMsgFlag` 变量。重置 `chooseFiles` 变量将清空图片选择面板中已选择的图片，重置 `sendMoreMsgFlag` 变量将再次隐藏图片选择面板。

需要注意的是，图片和文字可以存在于同一条评论中。开发者可以尝试发表几条带图片的评论。如果此时你点击评论中的图片，就会发现图片无法显示。关于这一点我们已在 7.9 节中提到过：在开发工具中是无法预览这些图片的。如果是在真机中，这些图片可以被正常预览。最新的 140600 版本修复了这个问题，现在可以在开发工具中预览上传的图片。

7.24 实现语音消息的发送

到目前为止，文字和图片评论的发送功能已全部完成，接下来我们学习如何发送语音评论。需要注意的是，语音消息的相关组件已在 7.10 节中实现，并在 7.12 节中实现了文字评论和语音评论之间的切换效果，大家可以回顾一下。

语音评论需要真机的支持，虽然在模拟器中有麦克风就可以实现录音并可以生成录音文件，但在后续的函数调用和播放上存在不少 bug。所以，下面的代码虽然在模拟器上不会报错，但无法实现录音效果。如果你不能够在真机上运行小程序，建议简单浏览下本节，了解内容即可。

发送语音评论的操作过程为：

- 切换到语音发送状态（点击最左侧的声音图标）。
- 长按“按住说话”这个按钮。
- 说话。
- 松开“按住说话”，语音消息自动发送。

要实现按住和松开这两个动作，我们需要使用小程序的 `touchstart` 和 `touchend` 事件。对于 `touchstart`，我们已注册了事件函数 `recordStart`；而对于 `touchend`，我们已注册了 `recordEnd`。下面实现这两个事件函数。



代码清单 7-51 实现 recordStart

post-comment.js

```
//开始录音
recordStart: function() {
  var that = this;
  this.setData({
    recodingClass: 'recoding'
  });
  //记录录音开始时间
  this.startTime = new Date();
  wx.startRecord({
    success: function (res) {
      //计算录音时长
      var diff = (that.endTime - that.startTime) / 1000;
      diff = Math.ceil(diff);

      //发送录音
      that.submitVoiceComment({ url: res.tempFilePath, timeLen: diff });
    },
    fail: function (res) {
      console.log(res);
    },
    complete: function (res) {
      console.log(res);
    }
  });
}
```

按住录音按钮后将执行 recordStart 函数。函数首先绑定了变量 recodingClass，这个变量将改变录音按钮的样式，使其变成正在录音的样式。

接着记录了当前录音开始的时间，并保存在 this 变量中。

接着调用 wx.startRecord 录音 API。wx.startRecord 只接受 3 个方法作为参数，分别是 success、fail 和 complete。

如果录音成功就执行 success，录音失败则执行 fail。无论录音成功还是失败，都将执行 complete。

发生以下两种情况将会结束录音：

- 当主动调用 wx.stopRecord 时。
- 如果没有主动调用 wx.stopRecord，那么在录音开始 1 分钟后自动结束录音。

在我们的 Orange Can 项目中并未处理录音 1 分钟自动结束的情况，开发者可根据自身需求来处理这种情况。



对于业务逻辑，我们需要在用户松开录音按钮时结束录音。下面编写 recordEnd 方法结束录音。

代码清单 7-52 结束录音

post-comment.js

```
//结束录音
recordEnd: function() {
    this.setData({
        recodingClass: "
    });
    this.endTime = new Date();
    wx.stopRecord();
}
```

在 recordEnd 方法中首先将按钮的样式还原，接着记录录音结束的时间，并将其保存在 this 变量中。最后调用 wx.stopRecord 方法结束录音。

当用户松开录音按钮后，代码将执行 recordStart 中的 success 方法或 fail 方法（complete 方法当然也会被执行）。

在 recordStart 的 success 方法中，我们首先计算语音时长，接着调用 submitVoiceComment 方法发送语音评论。在 post-comment.js 中增加以下代码来实现语音评论的发送。

代码清单 7-53 发送语音评论

post-comment.js

```
//提交录音
submitVoiceComment: function(audio) {
    var newData = {
        username: "青石",
        avatar: "/images/avatar/avatar-3.png",
        create_time: new Date().getTime() / 1000,
        content: {
            txt: "",
            img: [],
            audio: audio
        },
    };

    //保存新评论到缓存数据库中
    this.dbPost.newComment(newData);

    //显示操作结果
    this.showCommitSuccessToast();

    //重新渲染并绑定所有评论
```



```
this.bindCommentData();
```

```
}
```

发送语音评论的思路与发送文本、图片的思路几乎一样。首先新增一条评论数据，并将 audio 对象存入评论的 audio 属性中；然后调用 DBPost 的 newComment 方法将评论数据保存到缓存数据中；接着弹出操作结果提示；最后重新渲染评论列表。

语音不可以和文字、图片混合在一条评论中，只能单独作为一条评论。

需要注意的是，当前 130400 版本有一个奇怪的现象：在模拟器中，录音结束后既不会执行 wx.startRecord 的 success，也不会执行 fail，只会执行 complete 方法，即使 PC 接有麦克风也是如此。在真机中则可正常执行 success。

7.25 实现语音消息的暂停与播放

下面我们来实现评论列表中语音的播放与暂停功能。

语音评论的播放需要满足以下几个播放场景。假设有两条语音——A 语音和 B 语音，当点击 A 语音时：

- 如果 A 语音处于未播放状态，就开始播放 A 语音。
- 如果 A 语音处于暂停状态，就则继续播放 A 语音。

当点击 B 语音时：

- B 语音的行为同上述的 A 语音。
- 无论 A 语音处于何种状态，都将立刻被中断；被中断后，再次点击 A 语音，A 语音将重新开始播放。

下面我们来实现语音播放的功能。在代码清单 7-22 里，我们已经将 catchtap="playAudio" 这个事件注册到了语音评论的组件中，下面实现 playAudio 这个事件响应函数。

在 post-comment.js 中添加 playAudio 代码。

代码清单 7-54 编写 playAudio 方法

post-comment.js

```
playAudio: function (event) {
  var url = event.currentTarget.dataset.url,
      that = this;

  //暂停当前录音
  if (url === this.data.currentAudio) {
    wx.pauseVoice();
    this.data.currentAudio = ""
  }
}
```




```
//播放录音
```

```
else {
```

```
  this.data.currentAudio = url;
```

```
  wx.playVoice({
```

```
    filePath: url,
```

```
    complete: function () {
```

```
      //只有当录音播放完才会执行
```

```
      that.data.currentAudio = "";
```

```
    }
```

```
  });
```

```
}
```

```
}
```

在 `playAudio` 方法中，我们使用到了一个 `this.data.currentAudio` 变量，这个变量保存了当前正在播放的语音文件的 URL，同时它也将作为语音播放的控制开关。按照我们之前所约定的规则，对于这样的变量，我们在使用时最好在 `data` 中预先定义一下。

代码清单 7-55 定义 `currentAudio` 变量

post-comment.js

```
data: {
```

```
  // 控制使用键盘还是发送语音
```

```
  useKeyboardFlag: true,
```

```
  // 控制 input 组件的初始值
```

```
  keyboardInputValue: "",
```

```
  // 控制是否显示图片选择面板
```

```
  sendMoreMsgFlag: false,
```

```
  // 保存已选择的图片
```

```
  chooseFiles: [],
```

```
  // 被删除的图片序号
```

```
  deleteIndex: -1,
```

```
  //保存当前正播放语音的 URL
```

```
  currentAudio: ""
```

```
}
```

小程序提供了一个 `wx.playVoice` 方法，用于播放语音；同时提供了一个 `wx.pauseVoice` 方法，用于暂停语音播放；以及一个 `wx.stopVoice` 用于停止语音播放。

`wx.playVoice` 具有以下 4 个参数：

- `filePath` 需要播放的语音文件的文件路径。
- `success` 接口调用成功的回调函数。
- `fail` 接口调用失败的回调函数。
- `complete` 接口调用结束的回调函数（调用成功、失败都会执行）。

语音播放有一个特点：如果调用 `wx.pauseVoice` 暂停了语音播放，那么再次调用 `wx.playVoice` 播放同一个文件时就会从暂停处开始播放。如果想从头开始播放，需要先调用 `wx.stopVoice`。

我们来分析一下代码清单 7-54。

每次点击语音，`playAudio` 都将先从事件的 `event` 参数中获取要播放语音的 URL 地址 `url`。

接着判断该 `url` 与正在播放的语音地址 (`this.data.currentAudio`) 是否相同，如果相同就说明用户要暂停当前语音的播放，所以调用 `wx.pauseVoice` 方法来暂停语音播放。暂停语音播放后，请将 `this.data.currentAudio` 变量设置为空值。

如果 `url` 与 `this.data.currentAudio` 不同，就需要调用 `wx.playVoice` 来播放语音。

到底是播放新语音还是继续播放上次暂停的语音不需要用户来控制，`wx.playVoice` 自身会记录上一次播放语音的 `url`，它将自行做出比对：如果两次播放的 `url` 相同，且第一次播放的语音没有播放完（被暂停了），就将继续上一次语音的播放；如果第二次播放的语音与第一次不同，就直接播放新语音。

上面的逻辑强调的是第一次播放的语音没有播放完，如果已经播放完了，就将重头开始播放第一次的语音。

要特别注意 `wx.playVoice` 的 `complete` 函数，这个函数只有在语音完全播放完成后才会执行，如果暂停了播放，`complete` 函数是不会被执行的。同样，`wx.playVoice` 的 `success` 函数也是如此。

7.26 用户授权

小程序如果想使用某些特定的 API，是需要用户主动授权的。比如，当获取用户敏感信息时，以及调用 `wx.startRecord` 接口录音时，都需要用户主动授权，如图 7-17 和图 7-18 所示。

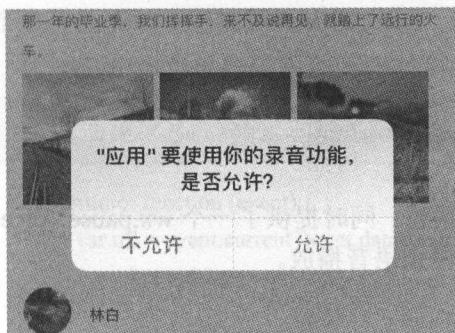


图 7-17 录音功能用户授权

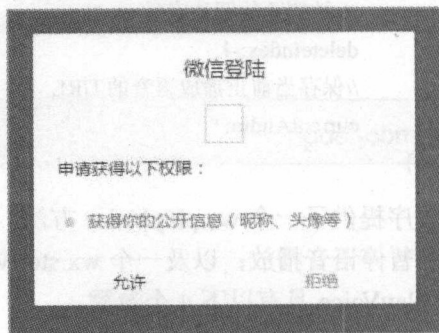


图 7-18 获取用户信息授权

在本项目中，如果用户第一次使用录音功能，就会弹出如图 7-16 所示的请求授权提示。当用户授权后，下次再使用录音功能时将不会再弹出这个提示框。

我们可以在开发工具中清除用户的授权数据，让授权提示框再次出现。

开发工具的侧边栏里有一项【缓存】按钮，如图 7-19 所示。

【清除工具授权数据】可以清除开发工具中用户授权数据缓存。

【清除手机授权数据】可以清除手机上的授权数据缓存。

清除缓存后，授权数据框将再次出现。

这里要提醒开发者，一定要处理用户拒绝授权的场景，否则程序有可能发生非常严重的错误。

清除数据缓存

清除文件存储

清除工具授权数据

清除手机授权数据

图 7-19 缓存管理菜单

7.27 解决真机运行时评论页面滑动卡顿的问题

在真机上运行 Orange Can，会发现评论页面在上下滑动时过程并不是那么的流畅，体验非常差。

在页面容器中加入以下 CSS 代码可以解决这个问题：

代码清单 7-56 解决卡顿的关键代码

post-comment.css

```
.comment-main-box {
  position: absolute;
  top: 0;
  left: 0;
  bottom: 100rpx;
  right: 0;
  overflow-y: auto;
  -webkit-overflow-scrolling: touch;
}
```

加粗部分代码是我们在 `comment-main-box` 类下新增的一段代码，加入这段代码后，`post-comment` 页面的滑动将非常流畅。

7.28 文章阅读计数功能

到目前为止，我们已经完成了文章的收藏、评论和点赞功能，接着完成最后一个文章计数功能。

每次进入 `post-detail` 页面时，当前文章的阅读数需要增加 1 次。在 `post-detail.js` 中增加一个阅读数加 1 的方法。

代码清单 7-57 阅读数加 1

post-detail.js

```
//阅读量+1
addReadingTimes:function(){
    this.dbPost.addReadingTimes();
}
```

addReadingTimes 方法中再一次使用到了 DBPost 中的方法。在 DBPost.js 中新增以下方法:

代码清单 7-58 阅读数加 1 的缓存操作方法

DBPost.js

```
//阅读数+1
addReadingTimes(){
    this.updatePostData('reading');
}
```

接着在 DBPost 类的 updatePostData 方法中增加一个 case 'reading'来处理阅读数加 1 的情况。下面给出 updatePostData 方法的最终代码:

代码清单 7-59 updatePostData 方法最终代码

DBPost.js

```
//更新本地的点赞、评论信息、收藏、阅读量
updatePostData(category, newComment) {
    var itemData = this.getPostItemById(),
        postData = itemData.data,
        allPostData = this.getAllPostData();
    switch (category) {
        case 'collect':
            //处理收藏
            if (!postData.collectionStatus) {
                //如果当前状态是未收藏
                postData.collectionNum++;
                postData.collectionStatus = true;
            } else {
                // 如果当前状态是收藏
                postData.collectionNum--;
                postData.collectionStatus = false;
            }
            break;
        case 'up':
            if (!postData.upStatus) {
                postData.upNum++;
                postData.upStatus = true;
            } else {
```




```

        postData.upNum--;
        postData.upStatus = false;
    }
    break;
    case 'comment':
        postData.comments.push(newComment);
        postData.commentNum++;
        break;
    case 'reading':
        postData.readingNum++;
        break;
    default:
        break;
    }
    allPostData[itemData.index] = postData;
    this.execSetStorageSync(allPostData);
    return postData;
}

```

加粗部分代码是新增代码。

编写完以上代码后，记得在 post-detail.js 中的 onLoad 方法中调用一下 addReadingTimes 方法。

代码清单 7-60 调用 addReadingTimes 方法

post-detail.js

```

onLoad: function (options) {
    var postId = options.id;
    this.dbPost = new DBPost(postId);
    this.postData = this.dbPost.getItemById().data;
    this.setData({
        post: this.postData
    })
    this.addReadingTimes();
}

```

加粗部分代码是新增代码。

完成以上代码后，每次点击进入页面的详情页阅读数都会加 1。

需要注意的是，本项目并没有实现实时刷新。进入某篇文章的详情页面后，再返回文章阅读列表页面，此时阅读列表中的阅读数并没有加 1，当我们刷新项目或者下次进入小程序时，文章列表的阅读数将会被更新。

第 8 章

背景音乐播放

本章我们将一起学习小程序的背景音乐播放 API，完整地实现了多个页面的背景音乐。如果开发者想开发一个类似于 QQ 音乐一样的音乐平台，那么本章将给你带来你所需要的基础知识。

8.1 显示音乐播放图标

本节我们来学习小程序的音乐播放相关 API。

我们尝试在每篇文章的详情页面上添加一个音乐播放的开关，这个开关位于文章详情页面头部图片的中部。

在 post-detail.wxml 中增加以下加粗部分代码：

代码清单 8-1 新增音乐播放图片

post-detail.wxml

```
<view class="container">
  <image class="head-image" src="{{post.postImg}}"/></image>

  <image catchtap="onMusicTap" class="music"
    src="{{isPlayingMusic? '/images/icon/wx_app_music_stop.png':
      '/images/icon/wx_app_music_start.png'}}">
  </image>

  <!--.....省略若干代码-->
</view>
```

接着在 post-detail.wxss 中添加以下 CSS 代码：

代码清单 8-2 添加音乐播放的 CSS 代码

post-detail.wxss

```
.music {
  width: 110px;
  height: 110px;
  position: absolute;
  left: 50%;
  margin-left: -51px;
  top: 180px;
  opacity: 0.9;
}
```

在 post-detail.js 的 data 变量中新增属性 isPlayingMusic 作为音乐播放状态的控制变量：

代码清单 8-3 新增 isPlayingMusic 控制变量

post-delal.js

```
data: {
  isPlayingMusic:false
}
```

保存代码后，详情页面将出现一个音乐播放的图片，如图 8-1 所示。



图 8-1 音乐播放图片

8.2 切换音乐播放图标

音乐播放图标将随着用户点击图标而发生状态的改变，这个改变类似于我们在第 7 章中处理语音播放，图标也将拥有播放和暂停两种状态。

我们先来实现简单的图片切换。

在 `post-detail.js` 中编写音乐图标的 `onMusicTap` 事件响应函数。

代码清单 8-4 onMusicTap 切换图标

post-detail.js

```
onMusicTap: function (event) {  
  this.setData({  
    isPlayingMusic: !this.data.isPlayingMusic  
  })  
}
```

保存后，点击音乐播放图片将在 `start` 和 `stop` 两种图标状态下切换。

8.3 背景音乐播放的特点

音乐播放有以下几个特点：

- 音乐播放不受页面关闭的影响，即使一个页面被 `unload` 掉，音乐依然会继续播放。所以在官方的 API 中音乐被称为背景音乐，所有音乐相关的 API 中也都会包含一个 “Background” 关键字。但我们之前学习的语音播放却没有这样的特性，当页面被 `unload` 后，音频播放将停止。

- 同时只能有一个后台音乐在播放，如果播放另外一首音乐，那么当前音乐将停止。
- 如果用户不主动关闭音乐，那么只有在退出小程序后音乐播放才会停止。关闭当前页面是不会影响小程序音乐播放的。
- 除了调用 MINA 框架的 API 来控制音乐播放，小程序在模拟器中还提供了一个总控开关来控制音乐的播放，如图 8-2 所示。

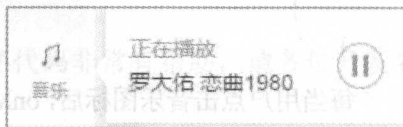


图 8-2 模拟器中的音乐总控开关

模拟器上的总控开关默认是不出现的，当播放一首音乐后，开发工具侧边栏上将出现【音乐】这个栏目。在 122100 版本之前，除了模拟器中有总控开关外，真机上也有一个总控开关。所谓总控开关，就是由系统提供的控制音乐播放和暂停的控制面板。在 130400 版本中，已无法在真机上找到这个总控开关。

- 一首音乐拥有 3 个主要属性：dataUrl(音乐的播放地址)、title(音乐标题)、coverImgUrl(音乐封面图 URL)。
- 歌曲只能是网络流媒体，不能播放本地音乐文件。比如，试图在小程序的文件夹里放置一首 mp3，但引用这个 mp3 的路径是无效的。

8.4 实现单页面背景音乐播放

我们在 onMusicTap 中调用 wx.playBackgroundAudio 和 wx.pauseBackgroundAudio 来实现背景音乐的播放与暂停效果。修改 post-detail.js 的 onMusicTap 方法如下：

代码清单 8-5 音乐播放

post-detail.js

```
onMusicTap: function (event) {
  if (this.data.isPlayingMusic) {
    wx.pauseBackgroundAudio();
    this.setData({
      isPlayingMusic: false
    })
  }
  else {
    wx.playBackgroundAudio({
      dataUrl: this.postData.music.url,
      title: this.postData.music.title,
      coverImgUrl: this.postData.music.coverImg
    })
    this.setData({
```

```

    isPlayingMusic: true
  })
}
}

```

每当用户点击音乐图标后, `onMusicTap` 都将判断当前是否有音乐正在播放 (`isPlayingMusic`)。如果当前有音乐正在播放, 就调用 `wx.pauseBackgroundAudio()` 函数暂停当前音乐播放, 并将音乐播放图标更换为暂停图标 (`isPlayingMusic` 状态变为 `false`)。如果当前没有音乐播放, 就调用 `wx.playBackgroundAudio(object)` 来播放一首音乐。

正如前面所讲, `wx.playBackgroundAudio` 接收 3 个参数: `dataUrl`、`title`、`coverImgUrl`。我们已经在 `data.js` 初始化文件中写好了这 3 个数据, 在这里只需要从 `this.postData` 中读取即可。`this.postData` 已经在页面的 `onLoad` 函数中设置过了。

保存并运行代码。在当前文章详情页面中, 音乐是可以正常播放与暂停的。除了在本页面可以播放音乐, 还可以切换到其他文章的详情页面中播放其他歌曲。当播放新歌曲时, 上一首歌将自动停止。

无论我们如何操作页面, 只要小程序不退出, 音乐就不会停止。

看似好像没什么问题了。真的是这样吗?

让我们做一下测试:

- (1) 进入文章 A 的详情页面, 点击音乐图标播放音乐。
- (2) 返回文章列表页面。
- (3) 随后再次进入文章 A 的详情页面。

这时留意一下音乐播放图标: 当前 A 音乐正在播放, 但音乐播放的图标却是未播放状态。为什么会出现这种情况? 原因在于, 当我们从 A 文章详情页面退出后, A 页面将会被 `unload` 掉, 所有属于页面的变量 (比如 `data`) 都将“消失”; 但我们之前讲过, 音乐播放是全局行为, 不会因为当前页面被 `unload` 掉了就停止播放。当再次进入 A 页面时, `isPlayingMusic` 变量将被初始化设置为 `false`, 而音乐却还在播放。这就造成了音乐播放图标状态不对的问题。

我们可以来验证一下, 当从 `post-detail` 子页面返回到 `post` 页面时是否会执行页面的 `unload` 函数。在 `post-detail.js` 中添加 `onUnload` 页面生命周期函数。

代码清单 8-6 测试 `post-detail` 页面是否 `unload`

`post-detail.js`

```

onUnload:function(event){
  console.log("page is unload");
}

```

当从当前 `post-detail` 页面点击左上角返回时, 将打印“page is unload”。这说明, 当子页面返回到父页面后, 子页面会被卸载。

在早前的版本中, 子页面返回到父页面后, 子页面的 `unload` 是有问题的。正确的 `unload` 行为是在 122100 版本中才修正的。Orange Can 项目在早期编写时由于子页面 `unload` 行为不正确 (没有完全清除子页面相关资源), 会造成大量的子页面“残留”在小程序中。当我们在小



程序中使用音乐监听函数时（后面会讲到音乐播放监听），不仅当前页面会触发监听函数，曾经打开过的所有子页面由于没有被正确卸载，都会去触发监听函数，这会给编码造成巨大的困扰。

理解子页面被卸载这个特性对于我们深入编写小程序代码非常有帮助，请各位开发者牢记。

一个简单的解决方案是，当我们从文章详情页面返回文章列表页面时，主动关闭音乐。关闭音乐需要使用 MINA 框架提供的 `wx.stopBackgroundAudio` 方法。

代码清单 8-7 关闭音乐播放

post-detail.js

```
onUnload: function (event) {
  wx.stopBackgroundAudio()
  this.setData({
    isPlayingMusic: false
  })
}
```

我们在 `post-detail` 页面的 `onUnload` 方法中主动关闭当前音乐播放，并设置 `isPlayingMusic` 状态为 `false`。

你也可以不设置 `isPlayingMusic` 的状态，因为页面关闭后，下次再进入页面时 `isPlayingMusic` 会自动被初始化为 `false`。

注意，本章节所提供的音乐播放地址不保证长期有效。如果音乐播放地址不可用，请自行寻找可播放的音乐地址，或者使用官方提供的示例音乐播放地址。

8.5 监听音乐播放

如果你有耐心听完一整首歌，就会发现一个小“bug”。当一首音乐播放完毕后，音乐图标应该恢复成未播放的状态，但是实际情况并不是这样。当一首音乐播放完毕后，音乐播放图片还是处于正在播放的状态。

思考一下这个问题，之前我们去控制音乐图标的状态时都是通过主动触发事件来实现的，但是音乐播放完成事件是没有办法主动触发的，它是被动的，是由 MINA 框架来触发的事件。

我们需要能够“监听”到音乐播放结束这个事件，并在事件函数中做处理。幸运的是，MINA 框架提供了 3 个“监听”函数，用来监听音乐播放的各种状态：

- `wx.onBackgroundAudioPlay(CALLBACK)` 监听音乐播放。
- `wx.onBackgroundAudioPause(CALLBACK)` 监听音乐暂停。
- `wx.onBackgroundAudioStop(CALLBACK)` 监听音乐停止。

我们使用 `wx.onBackgroundAudioStop(CALLBACK)` 来解决这个小“bug”。在 `post-detail.js` 的 `onLoad` 函数中新增一个 `setMusicMonitor` 方法。

代码清单 8-8 设置音乐播放监听

post-detail.js

```
setMusicMonitor: function () {  
  var that = this;  
  wx.onBackgroundAudioStop(function () {  
    that.setData({  
      isPlayingMusic: false  
    })  
  });  
}
```

在以上代码中，我们调用了 MINA 框架提供的 `wx.onBackgroundAudioStop(CALLBACK)` 方法来监听音乐播放停止这个事件。`wx.onBackgroundAudioStop(CALLBACK)` 接收一个回调函数作为参数，当音乐播放停止后，MINA 框架将主动调用这个 CALLBACK 回调函数。

我们在 CALLBACK 中更改 `isPlayingMusic` 这个状态变量为 `false`，从而将音乐图标恢复为未播放的状态。方法编写完成后，我们还需要在页面的 `onLoad` 函数中调用一下 `setMusicMonitor` 方法，这样监听函数才会生效。

代码清单 8-9 调用 `setMusicMonitor` 方法

post-detail.js

```
onLoad: function (options) {  
  var postId = options.id;  
  this.dbPost = new DBPost(postId);  
  this.postData = this.dbPost.getItemById().data;  
  this.setData({  
    post: this.postData  
  })  
  this.addReadingTimes();  
  this.setMusicMonitor();  
}
```

黑色加粗部分为新增加的代码。

8.6 全局变量与全局音乐播放

在之前的内容中，我们已经实现了单页面背景音乐的播放，但这种音乐播放体验并不是很好，用户不可能一直停留在某一个页面中。下面我们来实现全局音乐的播放。

其实，在之前的单页面背景音乐播放中全局播放音乐也是可以的，但前提条件是忽略音乐图标这个 bug。

我们可以深层次地分析一下出现 bug 的原因：`isPlayingMusic` 只是一个页面变量，而非全局变量，所以当页面销毁后页面变量丢失了，但音乐播放却还在进行。

那么，解决这个问题的思路就变成了需要找到一个全局变量来记录音乐播放的状态。这个全局变量和页面无关，不会因为页面的销毁而丢失。这样，变量的生命周期就可以和音乐播放的生命周期在同一个级别上了。

我们先来看看小程序的全局变量应该如何使用。

首先对比一下页面中的共享变量是如何设置的。页面的共享变量被设置在页面 Page 方法的 object 对象上，比如 data 就是 object 对象的一个属性。所以，我们在其他方法中才能够多次使用 this.data 的方式引用这个 data 对象。

页面的共享变量应该在页面中设置，所以全局共享变量自然应该在应用程序级别设置。

我们可以在 app.js 中添加以下代码，设置小程序的全局变量。

代码清单 8-10 添加全局变量

app.js

```
App({
  onLaunch: function () {
    var storageData = wx.getStorageSync('postList');
    if (!storageData) {
      var dataObj = require("data/data.js")
      wx.clearStorageSync();
      wx.setStorageSync('postList', dataObj.postList);
    }
  },
  globalData: {
    g_isPlayingMusic: false
  }
})
```

加粗部分代码是新添加的代码。我们在 App 的 object 对象中添加了一个 globalData 对象，这个对象用来记录整个项目的全局变量。

建议全局变量的变量名前统一加上一个“g_”前缀，以同普通变量区别开。

在 globalData 下添加一个 isPlayingMusic 变量，用来记录和管理音乐播放状态，初始化为 false。

设置了全局变量就要使用它，那么如何使用在 app.js 中定义的全局变量呢？下面尝试在 post-detail.js 中获取我们设置的 g_isPlayingMusic 变量。

在 post-detail.js 顶部增加以下代码：

代码清单 8-11 获取小程序 App 对象

post-detail.js

```
var app = getApp();
console.log(app)
Page({
  data: {
```

```

    isPlayingMusic: false
  },
  // .....省略若干代码
}

```

加粗部分为新增代码。注意，新添加的代码是位于 Page 函数之外的。

小程序提供了一个全局方法 `getApp()`，用于获取小程序的 App 对象。我们可以使用 `console.log` 输出这个 App 对象，如图 8-3 所示。

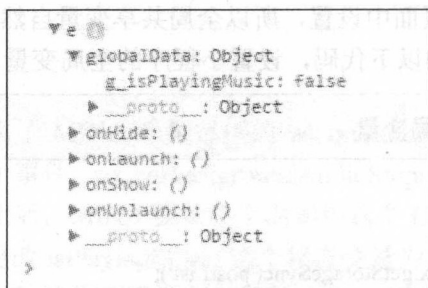


图 8-3 `getApp()` 获取到的小程序 App 对象

从图 8-3 中可以看到，我们在 `app.js` 中设置的 `globalData` 对象就在 App 对象下。这样我们在页面中就可以使用 `app.globalData` 来访问全局变量了。

下面我们来改写代码，让音乐播放支持全局播放。

首先屏蔽或者删除 `post-detail.js` 中的 `onUnload` 函数。这样当页面卸载时，音乐播放将不会停止。

接着，我们需要在每一次音乐播放状态改变时将改变的状态都更新保存到 `app.globalData.g_isPlayingMusic` 变量中。首先修改 `onMusicTap` 方法。

代码清单 8-12 修改 `onMusicTap` 方法

post-detail.js

```

onMusicTap: function (event) {
  if (this.data.isPlayingMusic) {
    wx.pauseBackgroundAudio();
    this.setData({
      isPlayingMusic: false
    })
    app.globalData.g_isPlayingMusic = false;
  }
  else {
    wx.playBackgroundAudio({
      dataUrl: this.postData.music.url,
      title: this.postData.music.title,
      coverImgUrl: this.postData.music.coverImg
    })
    this.setData({

```



```

        isPlayingMusic: true
    })
    app.globalData.g_isPlayingMusic = true;
}

```

加粗部分为新增加的代码。

同时，不要忘记，还需要在 setMusicMonitor 中更新全局音乐播放状态。

代码清单 8-13 修改 setMusicMonitor 方法

post-detail.js

```

setMusicMonitor: function () {
    var that = this;
    wx.onBackgroundAudioStop(function () {
        that.setData({
            isPlayingMusic: false
        })
        app.globalData.g_isPlayingMusic = false;
    });
}

```

加粗部分为新增代码。

以上两处改动保证了每次音乐播放状态改变时 g_isPlayingMusic 全局变量都会得到相应的更新。下面来使用 g_isPlayingMusic 变量。

每次在进入 post-detail 页面时，我们都应该读取 app.globalData.g_isPlayingMusic 的值，根据这个变量值来决定音乐播放图标的显示状态。在 post-detail.js 中添加一个初始化音乐图标状态的方法。

代码清单 8-14 初始化音乐播放图标状态

post-detail.js

```

initMusicStatus(){
    this.setData({
        isPlayingMusic: app.globalData.g_isPlayingMusic
    })
}

```

同时，在页面的 onLoad 函数中调用 initMusicStatus 方法。

代码清单 8-15 调用 initMusicStatus

post-detail.js

```

onLoad: function (options) {
    var postId = options.id;
    this.dbPost = new DBPost(postId);
    this.postData = this.dbPost.getPostItemById().data;
    this.setData({

```

```

        post: this.postData
    })
    this.addReadingTimes();
    this.setMusicMonitor();
    this.initMusicStatus();
}

```

加粗部分为新增代码。

保存并运行代码后发现，在 A 文章播放音乐，退出 A 页面后再进入 A 页面，音乐播放的图标仍是正在播放的状态。

我们在多“逛一下”各个页面，保持音乐播放的状态，随便进入其他任何文章详情页面就会发现问题。所有文章的音乐播放图标都变成了正在播放状态。事实上，我们只是播放了 A 文章的音乐，这是不对的。

如何解决这个问题呢？

除了利用 `app.globalData.g_isPlayingMusic` 变量记录音乐播放的状态外，我们还需要一个变量来记录当前正播放的是哪首歌曲，以免其他没有播放音乐的文章的音乐图标状态也发生改变。

在 `app.js` 中的 `globalData` 下添加一个 `g_currentMusicPostId`。

代码清单 8-16 新增全局变量保存正播放音乐的 id 号

app.js

```

globalData: {
  g_isPlayingMusic: false,
  g_currentMusicPostId: null
}

```

对于每首音乐，可以使用音乐所属文章的 id 号作为其身份编号。`g_currentMusicPostId` 将保存这个文章的 id 号，表示当前正在播放音乐的 id 号。

在 `post-detail.js` 页面的 `onMusicTap` 方法中新增一段代码，用来记录当前正在播放音乐的 id 号。

代码清单 8-17 保存音乐 id 号

post-detail.js

```

onMusicTap: function (event) {
  if (this.data.isPlayingMusic) {
    //暂停音乐播放
    wx.pauseBackgroundAudio();
    this.setData({
      isPlayingMusic: false
    })
    app.globalData.g_isPlayingMusic = false;
  }
  else {

```




```
//播放音乐
wx.playBackgroundAudio({
  dataUrl: this.postData.music.url,
  title: this.postData.music.title,
  coverImgUrl: this.postData.music.coverImg
})
this.setData({
  isPlayingMusic: true
})
app.globalData.g_isPlayingMusic = true;
app.globalData.g_currentMusicPostId = this.postData.postId;
}
```

加粗部分是新增代码。

有了当前正在播放音乐的 id 号，我们就可以在进入文章详情页面时准确判断音乐图标的显示状态了。

修改 post-detail 页面的 initMusicStatus 方法，加入对 app.globalData.g_currentMusicPostId 全局变量的判断。

代码清单 8-18 修改 initMusicStatus 方法

post-detail.js

```
initMusicStatus() {
  var currentPostId = this.postData.postId;
  if (app.globalData.g_isPlayingMusic &&
    app.globalData.g_currentMusicPostId === currentPostId) {
    // 如果全局播放的音乐是当前文章的音乐，就将图标状态设置为正在播放
    this.setData({
      isPlayingMusic: true
    })
  }
  else {
    this.setData({
      isPlayingMusic: false
    })
  }
}
```

上述代码中的 if else 条件语句将确保只有当音乐正在播放且正在播放的音乐是当前文章的音乐时，当前文章的音乐图标才会被设置为正在播放状态。

保存代码后，无论点击切换或者是跳出当前页面再进入页面，音乐图标都将正常显示。

8.7 音乐总控开关

在完成以上代码后，背景音乐的播放就算完成了。但在 8.3 节中提到过，在模拟器中还有一个音乐总控开关，如图 8-4 所示。

点击这个开关同样可以实现音乐的播放与暂停操作。我们可以点击一下这个按钮，音乐确实随着开关的操作不断地暂停和播放，但文章详情页面的音乐图标却不能跟随操作开关的操作而变化。

在 122100 版本之前这是一个很大的问题，因为在真机上是有音乐总控开关的，如果用户使用音乐总控开关来操作音乐，那么文章详情页面的音乐图标就会出现状态错乱的问题。但在目前的 130400 版本中，真机上的音乐播放开关“消失了”。之前当退出小程序时，音乐播放也不会停止，在微信的首页会出现一个悬浮条，点击这个悬浮条可以进入如图 8-5 所示的页面。

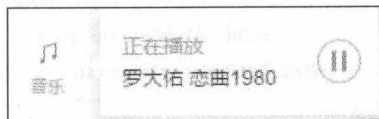


图 8-4 音乐总控开关

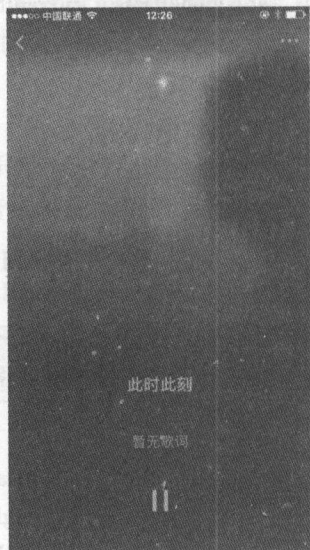


图 8-5 真机上的音乐总控

在这个页面同样可以暂停和播放音乐。同时，这个页面的背景图就是我们在调用 `wx.playBackgroundAudio` 时所设置的 `coverImgUrl` 参数。取消真机上的音乐播放开关后，这个页面再也无法打开，所以设置封面图是没有效果的，因为你无法看到这个封面图。注意，以上结论仅限于在当前的 130400 版本中，至于以后的版本会不会在真机中又开放这个音乐播放开关就不得而知了。

至少在目前情况下你无须关注音乐总控开关对于文章详情页面中音乐图标状态的影响，将代码编写到目前为止的状态就可以了，总控开关是无法在真机上影响图标状态的。

作为一个扩展，我们还是要来纠正这个“错误”，让项目在模拟器中也可以正确响应音乐总控开关的操作。如果你不感兴趣，可以直接跳过这一节的内容。

响应总控开关事件的思路依然是使用音乐监控 API，开发者可回顾一下 8.5 节中所讲述的 3 个音乐状态监听函数。我们之前已经在 `setMusicMonitor` 方法中使用过 `wx.onBackgroundAudioStop` 方法。要解决总控开关的问题，还需要使用另外两个监听函数，分别来响应总控开关暂停音乐和总控开关播放音乐。

修改 `setMusicMonitor` 方法。

代码清单 8-19 添加总控开关事件监听函数

post-detail.js

```

setMusicMonitor: function () {
    var that = this;
    wx.onBackgroundAudioStop(function () {
        that.setData({
            isPlayingMusic: false
        })
        app.globalData.g_isPlayingMusic = false;
    });

    wx.onBackgroundAudioPlay(function (event) {
        // 只处理当前页面的音乐播放
        if (app.globalData.g_currentMusicPostId === that.postData.postId) {
            that.setData({
                isPlayingMusic: true
            })
        }
        app.globalData.g_isPlayingMusic = true;
    });

    wx.onBackgroundAudioPause(function () {
        // 只处理当前页面的音乐暂停
        if (app.globalData.g_currentMusicPostId === that.postData.postId) {
            that.setData({
                isPlayingMusic: false
            })
        }
        app.globalData.g_isPlayingMusic = false;
    });
}

```

代码加粗部分为新增代码。处理逻辑非常简单，同 `wx.onBackgroundAudioStop` 监听音乐停止一样，我们分别使用 `wx.onBackgroundAudioPlay` 和 `wx.onBackgroundAudioPause` 方法来监听音乐的播放与暂停事件，从而设置文章页面的音乐图标状态，以使音乐图标的状态和音乐播放的状态保持一致。

需要特别注意的是，一定要判断总控开关播放和暂停的音乐是不是当前页面的音乐。不加入这个判断，将会导致以下 bug：当前音乐播放的是文章 A 的音乐，但用户正停留在文章 B 的页面，此时操作总控开关，明明操作的是文章 A 的音乐，却导致文章 B 页面的音乐播放图标状态改变了。这显然是不符合逻辑的。

8.8 显示音乐的封面图片

虽然小程序自己的播放器不能够显示音乐封面图片，但是我们初始化数据中的音乐封面图也不能浪费。我们尝试自己在小程序里显示这个音乐封面图。

下面实现一个小功能：当用户点击播放音乐后，不仅音乐播放图标会改变，文章详情页面的文章图片也将随之切换为音乐的封面图；而当音乐播放暂停时，这个图片又将切换回文章的图片。

音乐封面图的显示效果如图 8-6 所示。



图 8-6 音乐播放时的封面图

就我们现在的代码完成度，要实现上述切换功能非常简单，只需要修改一句代码即可。

修改 `post-detail.wxml` 页面中的 `<image class="head-image" src="{{post.postImg}}">` `</image>` 为以下代码：

代码清单 8-20 显示音乐封面图

post-detail.wxml

```
<image class="head-image"
src="{{isPlayingMusic?post.music.coverImg:post.postImg}}">
</image>
```

其中的 `isPlayingMusic` 早已在之前的 `post-detail.js` 文件中编好了切换逻辑，这里直接使用就可以了。

从这一段代码的修改上来看，数据绑定是不是相当简洁、方便？



第 9 章

丰富文章页面

本章是 Orange Can 项目“文章”部分的收尾和完善，其中包括小程序很重要的分享功能以及 animation 动画的使用。

9.1 将页面分享给朋友和微信群

在内测和公测期间，小程序是没有任何分享能力的，但就在小程序临近正式开放时，微信放出了这个分享功能。我们首先要提醒的是不同于公众号的分享，微信小程序只能分享给好友和群聊，不能分享到朋友圈。


在 122100 版本中，小程序在右上角增加了一个如图 9-1 所示的  按钮。在模拟器中点击这个按钮将出现如图 9-2 所示的底部菜单栏。



图 9-1 右上角的按钮

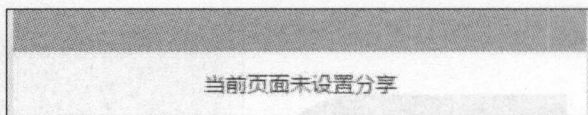


图 9-2 在模拟器中点击右上角按钮

如果我们没有在当前页面调用分享的 API，那么点击右上角的按钮将出现“当前页面未设置分享”的提示。

注意，分享按钮是页面的行为，而不是应用程序的行为，每个页面都可以调用分享 API 并设置自己的分享参数。

MINA 框架提供了一个用于设置页面分享的 API: `onShareAppMessage`。注意，这个 API 是一个页面方法，而不是我们前面调用的 `wx` 类方法（`wx.playBackgroundAudio` 等）。它和 `onLoad`、`onShow` 等一样，属于页面级别的方法。所以，你不可以自定义分享按钮，只能使用小程序页面右上角的按钮进行分享。

`onShareAppMessage` 方法必须返回一个 object 对象，这个对象可以包含以下 3 个属性：

- `title` 设置分享标题，如果忽略这个参数，就将默认 `title` 为当前小程序的名称。
- `desc` 分享描述，如果忽略这个参数，就将默认 `title` 为当前小程序的名称。
- `path` 分享路径，当前页面 `path` 是以 / 开头的完整路径。注意，这个 `path` 同 `wx.NavigateTo` 一样，可以在 `path` 后面添加形如 `id=2&mid=3` 的参数。获取方式同样是在页面的 `onLoad` 函数的参数中获取。


现在我们来尝试在 `post-detail` 页面中加入分享功能，在 `post-detail.js` 中增加以下方法：

代码清单 9-1 定义页面分享函数

post-detail.js

```
onShareAppMessage:function(){
  return {
    title:this.postData.title,
    desc:this.postData.content,
    path:"/pages/post/post-detail/post-detail"
  }
}
```



在 `post-detail.js` 中加入以上函数后, 点击右上角的  按钮, 页面底部将不再显示图 9-2 中所示的【当前页面未设置分享】, 取而代之的是可以点击的分享菜单, 如图 9-3 所示。

点击分享后将弹出分享确定界面, 如图 9-4 所示。

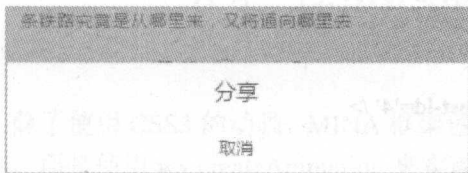


图 9-3 设置分享后的底部弹出菜单

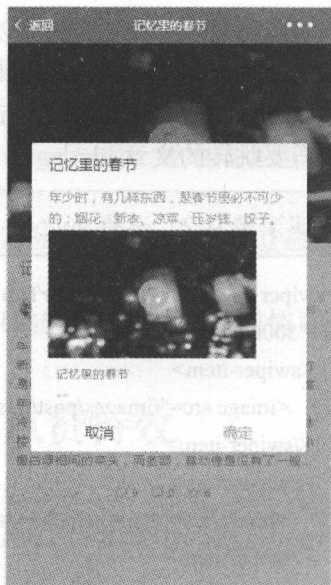


图 9-4 弹出分享内容

需要特别注意的是, 分享图片是不能自定义的, MINA 框架会将当前页面从顶部开始、高度为 80% 屏幕宽度的图像作为分享图片。所以, 对于我们需要分享的页面, 请好好设计页面的顶部部分, 以免分享出去的图片不美观。

官方文档对于 `onShareAppMessage` 函数有一段这样的描述: “只有定义了此事件处理函数, 右上角菜单才会显示“分享”按钮”。

这段描述极易让人误以为如果不定义分享函数 `onShareAppMessage`, 右上角的三个小点就不会出现, 从而实现隐藏右上角按钮的效果。

事实上, 并不是像字面意思这么理解。官方文档的正确解读应该是: 只有定义了 `onShareAppMessage` 处理函数, 从右上角按钮打开的菜单才有“分享”按钮。无论如何, 右上角的按钮是无法隐藏或者取消的。

右上角按钮还有一个重要作用, 就是点击右上角后在弹出的菜单中会有一个“显示在聊天顶部”选项。点击“显示在聊天顶部”将折叠小程序, 并返回微信聊天列表页面。当再次点击被折叠的小程序时, 将重新进入小程序。

这个功能将帮助用户在使用小程序时临时切换到微信消息界面进行消息查看和回复。这个功能只有在真机上才会出现, 开发工具中是没有这个体验功能的。

虽然小程序目前只能分享给朋友和群聊, 但其意义依然非常重大, 至少给了开发者一个线上推广和传播的机会。在目前小程序发展初期, 很多小程序的推广都是依靠微信群聊在传播。

9.2 从 swiper 组件跳转到文章详情页面

截至目前, `post-detail` 详情页面就已经完成了全部功能, 但是在 `post` 文章列表页面还有一

点小小的功能需要补充,既然可以点击文章列表的文章跳转到文章详情页面,那么文章列表顶部的 swiper 组件也应该能够点击跳转。

首先对 post.wxml 页面的 swiper 组件做一些小小的修改,在每个 swiper 组件的 image 元素上设置需要跳转的文章 id 号。

代码清单 9-2 设置 swiper-item 元素的文章 id 号

post.wxml

```
<swiper catchtap="onSwiperTap" vertical="{{false}}" indicator-dots="true" autoplay="true"
interval="5000" circular="true">
  <swiper-item>
    <image src="/images/post/post-1@text.jpg" data-post-id='3' />
  </swiper-item>
  <swiper-item>
    <image src="/images/post/post-2@text.jpg" data-post-id='4' />
  </swiper-item>
  <swiper-item>
    <image src="/images/post/post-3@text.jpg" data-post-id='5' />
  </swiper-item>
</swiper>
```

加粗部分代码为新增代码。注意,该 id 号必须是已存在的文章 id 号,否则跳转后无法获取文章详细信息。

按照一般的思路,跳转到文章详情页面需要在每个 swiper-item 组件上都注册一个 tap 事件,从而保证点击每一张图片都可以响应该事件。这样做当然是可以的,但我们设想一下,如果 swiper 组件下有十几个元素呢?这样一个个地去绑定事件是不是太麻烦了?

这里使用之前我们讲到的冒泡事件,不在每个 swiper-item 的 image 上注册事件,而只是在 swiper 上注册一个 onSwiperTap 事件。无论点击哪个 swiper-item 的 image,点击事件都将通过冒泡机制传递到 swiper-item 的父节点 swiper 上。所以,我们只需要在 Swiper 组件上捕获这个点击事件,无须在每个子元素上监听点击事件。

在 post.js 中编写事件响应函数 onSwiperTap。

代码清单 9-3 编写 onSwiperTap 函数

post.js

```
onSwiperTap: function (event) {
  var postId = event.target.dataset.postId;
  wx.navigateTo({
    url: "post-detail/post-detail?id=" + postId
  })
}
```

代码非常简单,思路就是获取文章 id 号后再通过 wx.navigateTo 导航跳转到 post-detail 文章详情页面。



需要注意的是,在获取文章 id 号时,我们并不是使用的 `event.currentTarget`,而是使用的 `event.target`。在冒泡事件中, `target` 指的是事件最开始被触发的元素,而 `currentTarget` 指的是捕获事件的元素。放在我们的代码中, `target` 指的是 `image` 元素,而 `currentTarget` 指的是 `swiper` 元素。点击 `swiper` 时实际上点击的是 `image` 组件,事件由 `image` 一级一级地传递到 `swiper` 组件中,最后被我们注册在 `swiper` 组件上的 `onSwiperTap` 捕获。

只有在 `image` 元素上才设置有 `postId`,从 `currentTarget(swiper)` 元素中是无法获取到 `postId` 的,所以我们必须使用 `event.target` 来获取 `postId`。

保存并运行代码,发现点击 `swiper` 组件的不同图片可以跳转到对应的文章详情页面。

9.3 使用小程序动画实现点赞特效

除了使用 CSS3 的动画,MINA 框架也提供了一组动画 API。在本节中,我们不使用 CSS3 动画,而是使用 `wx.createAnimation` 来实现点赞动画效果。

学习小程序的动画,首先要明白下面几个概念:

- 动画实例
- 动画组
- 动画方法
- 动画队列

在使用小程序动画前必须先创建一个动画实例,创建动画实例的方法为: `wx.createAnimation(object)`。该方法的 `object` 参数会接受一些属性,用于定义动画的具体执行参数:

- `duration` 数值整型,描述动画的持续时间,单位为 `ms` (毫秒),默认值为 `400`。
- `timingFunction` 字符串,定义动画的效果,默认值为 `"linear"`,有效值为 `"linear"` `"ease"` `"ease-in"` `"ease-in-out"` `"ease-out"` `"step-start"` `"step-end"`。
- `delay` 数值整型,表示动画延迟多少毫秒执行,单位为毫秒,默认值为 `0`,表示不延迟。
- `transformOrigin` 字符串类型,设置 `transform-origin`,默认为 `"50% 50% 0"`。

以上参数都为非必填参数,它们都有自己的默认值。

当我们使用 `wx.createAnimation` 创建一个动画实例 `animation` 后,可以对这个实例设置多个动画组,而每个动画组中会有若干个动画方法;组与组之间使用 `step()` 方法来分隔,多个组形成一个链式的调用队列,这个链式队列就是我们所说的动画队列。看一下下面的演示代码会比较清楚地理解以上几个概念。

代码清单 9-4 动画演示代码

```
//创建一个动画实例
```

```
var animation = wx.createAnimation({
  timingFunction: 'ease-in-out'
```

```

    })
    // 设置动画组（以 step()分隔），每个队列
    animation.scale(2,2).rotate(45).step().translate(30).step()

```

在以上代码中，首先我们创建了一个动画实例 `animation`，并设置这个 `animation` 的特效为 `ease-in-out`。接着我们对这个动画实例设置了两个动画组，它们之间使用 `step()` 作为分隔。第一组动画包含两个动画方法 `scale` 和 `rotate`（缩放和旋转）；第二组动画包含一个动画方法 `translate`（偏移）。

我们在这里提出一个问题，以上两个动画组以及每个动画组里的动画方法执行顺序是什么？`scale` 与 `rotate` 是同时执行还是先执行 `scale` 后执行 `rotate`？`rotate` 和 `translate` 又是谁先执行？毕竟是动画，每个动画的执行先后顺序是非常重要的。

请开发者牢记：同一组中的动画方法会同时执行，但动画组必须是先后执行。也就是说，一组动画先执行完成后，后面的动画才能执行，它们是 `one by one` 地执行。

用以上理论来解释以下示例代码中动画执行的先后顺序：`scale` 和 `rotate` 会同时执行，而 `translate` 必须等 `scale` 和 `rotate` 执行完成后才会执行。

还有一个问题，可不可以在不同的动画组中设置不同的动画效果参数？答案是可以的。每个 `step` 方法都可以接受一个 `object` 对象：可以传入一个跟 `wx.createAnimation()` 一样的配置参数，用于指定当前组动画的配置。

代码清单 9-5 step 接受动画配置

```

//创建一个动画实例
var animation = wx.createAnimation({
  timingFunction: 'ease-in-out'
})
// 设置动画组（以 step()分隔），每个队列
animation.scale(2, 2).rotate(45).step().translate(30).step({ duration: 1000 })

```

注意黑色加粗部分的代码，这将改变第二组动画 `translate` 的执行效果。

常见的动画方法有哪些？截至 130400 版本，小程序提供了 6 类动画方法：

- 样式 `opacity`、`backgroundColor`、`width`、`height`、`top`、`left`、`bottom`、`right`。
- 旋转 `rotate`、`rotateX`、`rotateY`、`rotateZ`、`rotate3d`。
- 缩放 `scale`、`scaleX`、`scaleY`、`scaleZ`、`scale3d`。
- 偏移 `translate`、`translateX`、`translateY`、`translateZ`、`translate3d`。
- 倾斜 `skew`、`skewX`、`skewY`。
- 矩阵变形 `matrix`、`matrix3d`。

以上动画的参数及使用方法请参考官方文档。动画属于实践性非常强的知识点，在这里用文字去罗列每个动画的使用方法效果并不好，反而会产生各种误解和歧义。如果要详细讲解动画知识，可以写一本书了；如果简单复制官方的示例文档，那不如开发者自己去查看官方的 `demo` 示例。所以强烈建议，如果你正在研究动画的使用方法，就请创建一个动画实例，根据



我们所讲的动画理论知识不断地变化动画实例参数、动画方法等，亲自看看都有什么不同的效果。

笔者在这里也强烈建议各位开发者，对于像动画这类实践性很强的知识点不要“死记硬背”。当你使用的时候再来查找这些动画方法，效果会更好。

代码清单 9-4 和代码清单 9-5 都是在定义动画，那么如何使用动画呢？

不用着急，我们现在拿 Orange Can 项目中的实例来学习如何定义和使用动画。还记得在 post-detail 页面中有一个点赞的按钮（心形图标）？我们用小程序动画来实现点赞的特效：当点击按钮时，心形图标会有一个渐渐放大随后又缩小的效果。回忆一下在本节之前的文字中介绍的动画基本概念与如何定义动画。下面我们来实践一下。

在 post-detail.js 中新增一个方法。

代码清单 9-6 创建 animation 实例

post-detail.js

```
setAniation: function () {
  //定义动画
  var animationUp = wx.createAnimation({
    timingFunction: 'ease-in-out'
  })

  this.animationUp = animationUp
}
```

以上代码定义了一个 setAniation 方法，接着我们在 post-detail.js 的 onLoad 方法中使用 setAniation 方法。

代码清单 9-7 调用 setAnimation 方法

post-detail.js

```
onLoad: function (options) {
  var postId = options.id;
  this.dbPost = new DBPost(postId);
  this.postData = this.dbPost.getItemById().data;
  this.setData({
    post: this.postData
  })
  this.addReadingTimes();
  this.setMusicMonitor();
  this.initMusicStatus();
  this.setAniation();
}
```

加粗部分为新增代码。



接着我们修改 onUpTap 这个点赞按钮的事件响应函数。如果你的点赞事件响应函数没有按照本书命名，那么请找到自己的点赞响应函数并修改。

代码清单 9-8 定义动画方法并使用动画

post-detail.js

```
onUpTap: function (event) {
  var newData = this.dbPost.up();

  this.setData({
    'post.upStatus': newData.upStatus,
    'post.upNum': newData.upNum
  }),

  this.animationUp.scale(2).step();
  this.setData({
    animationUp: this.animationUp.export()
  })
  setTimeout(function () {
    this.animationUp.scale(1).step();
    this.setData({
      animationUp: this.animationUp.export()
    })
  }.bind(this), 300);
}
```

黑色加粗部分代码是新增代码。

使用动画的关键有以下两点：

- 必须调用动画实例 animationUp 的 export 方法导出动画。
- 将 export 方法导出的动画绑定到动画需要作用的 wxml 组件上。

注意，当调用动画实例 animationUp 的 export 方法后，animationUp 上所设置的动画方法将被清空。

以上代码中，我们对动画实例 animationUp 做了两次设置和调用：第一次设置 scale 动画方法让图标先放大，随后调用 step 方法表示这组动画完成。接着调用 export 方法导出动画，并做数据绑定更新，这将导致点赞图标被放大。

第二次设置 scale 方法让图标恢复到原状，同样再次调用 step 和 export 并做数据绑定更新，这将导致点赞图标还原。

注意，第一次调用 export 后，animationUp 动画实例会被清空，所以第二次设置 animationUp 时不会受第一次动画方法的影响。

在执行第二次动画时，我们使用 setTimeout 让缩小的动画效果延迟 300 秒再执行。

既然我们在代码中使用了数据绑定，就必须在 wxml 中绑定这个动画，这样点赞动画才能够正常执行。



修改 post-detail.wxml 中设置点赞图标的 image 组件代码。

代码清单 9-9 绑定动画

post-detail.wxml

```
<view class="tool-item" catchtap="onUpTap" data-post-id="{{post.postId}}">
  <image animation="{{animationUp}}" wx:if="{{post.upStatus}}"
    src="/images/icon/wx_app_liked.png" />
  <image animation="{{animationUp}}"
    wx:else src="/images/icon/wx_app_like.png" />
  <text>{{post.upNum}}</text>
</view>
```

黑色加粗部分是我们新增加的代码，作用很简单——接受动画的数据绑定。注意，需要同时绑定已点赞和未点赞两种状态的图标。

编写完以上代码后，保存代码并运行，点击点赞图标将出现先放大再缩小的动画效果。

有些开发者可能会提出质疑，为什么要设置两次动画效果？之前不是说过，动画队列中的不同组动画将依次执行吗？理论上，确实可以通过设置两个动画组来实现先放大再缩小的效果。但截至 130400 版本，小程序动画存在以下已知 bug：iOS/Android 6.3.30，通过 step() 分隔动画，只有第一步动画能生效。

以上说明来自于官方文档，我们在模拟器上测试后，发现确实只有第一个分组的动画可以被执行，而第二组动画是无法执行的。所以，我们采用了以上代码的编写方法来实现先放大再缩小的动画效果。

笔者在之前编写 post-comment 评论页面时也向大家介绍过 CSS3 动画在小程序中的编写方法，这说明小程序依然是支持 CSS3 动画的。

CSS3 是 Web 标准且没有特别明显的 bug，如果你已经能够很熟练地使用 CSS3 动画，就请在小程序中优先使用 CSS3 动画。熟练使用 CSS3 动画对我们编写 Web 项目也是有好处的，毕竟 CSS3 在很多地方都可以使用，但是小程序动画却只能在小程序中使用。但 CSS3 动画也有一定的风险，CSS3 动画种类众多，小程序不一定全都支持，我们无法一一验证。

究竟如何抉择还请开发者自行考虑。

第 10 章

电 影

本章我们编写了一个类似“豆瓣影评”的小功能，所有数据来自于豆瓣开放的 API。通过编写这部分的功能，我们将学习如何使用 `wx.request` 获取真实的网络数据，并将这些数据“填充”到小程序中。

此外，本章中最重要的内容是多层嵌套模板的使用技巧，多层嵌套模板的优势将在本章中得到体现。

此外本章还指出了很多小程序的“坑”，希望可以帮助开发者节约宝贵的时间。

10.1 小程序的 tab 选项卡

从本章开始，我们将着手编写 Orange Can 项目的电影部分。电影部分同文章部分属于同一级别，我们在本章的开始部分使用小程序提供的 tab 选项卡来实现电影模块和文章模块的切换。tab 选项卡的效果如图 10-1 所示。

需要注意的是，我们不需要自己编写代码实现 tab 选项卡。小程序提供了现成的 tab 选项卡，我们只需要在 app.json 中配置一些参数即可实现 tab 选项卡的效果。

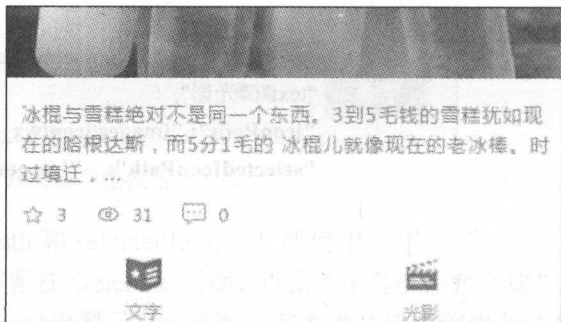


图 10-1 tab 选项卡效果

tab 选项卡的配置是通过 app.json 文件中的 tabBar 选项来实现的。在配置 tab 选项卡之前，我们新建一个页面——movie 页面（位于 pages/movie 目录下）。或者直接在 app.json 的 pages 选项下新增 pages/movie/movie 并保存，这将直接在 pages/movie 目录下新建 movie 页面的 4 个页面文件。

接着配置文章和电影页面的 tab 选项卡，代码如下：

代码清单 10-1 配置项目 tab 选项卡

app.json

```
{
  "pages": [
    "pages/welcome/welcome",
    "pages/post/post",
    "pages/post/post-detail/post-detail",
    "pages/post/post-comment/post-comment",
    "pages/movie/movie"
  ],
  "window": {
    "navigationBarBackgroundColor": "#4A6141"
  },
  "tabBar": {
    "borderStyle": "white",
    "selectedColor": "#4A6141",
    "color": "#333",
    "backgroundColor": "#fff",
    "position": "bottom",
    "list": [
    ]
  }
}
```



```

    "pagePath": "pages/post/post",
    "text": "文字",
    "iconPath": "images/icon/wx_app_news.png",
    "selectedIconPath": "images/icon/wx_app_news@HL.png"
  },
  {
    "pagePath": "pages/movie/movie",
    "text": "光影",
    "iconPath": "images/icon/wx_app_movie.png",
    "selectedIconPath": "images/icon/wx_app_movie@HL.png"
  }
]
}

```

黑色加粗部分是新增的代码。tabBar 配置项有以下几个属性：

- **color** 未选中时的 tab 选项卡文字颜色。
- **selectedColor** 选中时 tab 选项卡文字颜色。
- **backgroundColor** tab 选项卡背景颜色。
- **borderStyle** tab 选项卡上边框的颜色，注意它只支持 **black** 和 **white** 两个取值，默认是 **black**。
- **list** tab 选项卡列表，是一个数组，接受一组 **object** 对象，我们在后面会具体给出每个对象的属性。
- **position** 可选值有 **bottom** 和 **top**，默认为 **bottom**，指定选项卡位于底部还是顶部。

再来具体看看 **list** 这个数组。**list** 数组的每一项都是一个 **object** 对象，每个 **object** 对象代表一个 tab 选项，最少必须有两个 tab 选项，而最多只能有 5 个 tab 选项。tab 选项卡出现的顺序由数组中 **object** 的顺序来决定。**object** 对象包含以下几个属性：

- **pagePath** 每个 tab 选项的页面路径。注意，用于 **pagePath** 的路径必须预先已在 **app.json** 的 **pages** 中定义。
- **text** tab 选项卡上出现的文字。
- **iconPath** tab 选项卡上的图片路径，图片大小限制为最大 40KB，建议尺寸为 81px × 81px。
- **selectedIconPath** 选中是的图片路径，图片大小限制为最大 40kb，建议尺寸为 81px × 81px。

这里要特别注意，对于 **pagePath**、**iconPath** 和 **selectedIconPath** 这几个路径，一定不要以 **“/”** 开头。即使它们看起来是绝对路径也不要再在路径前面加 **“/”**。在 **pagePath** 前面加 **“/”** 将导致错误。如果在 **iconPath** 前加 **“/”**，虽然在模拟器中不会出现问题，但将项目在真机上预览时（在开发工具的“项目”里点击“预览”时）开发工具将报如图 10-2 所示的错误。





图 10-2 以 “/” 开头将导致报错

所以，建议开发者不要在 `pagePath`、`iconPath` 和 `selectedIconPath` 前使用 “/” 开头。

此时，我们保存并运行代码，会发现页面停留在 `welcome` 页面，点击“开启小程序之旅”，页面没有反应。如果在 `welcome.js` 的 `navigateTo` 中设置了 `fail` 函数，点击“开启小程序之旅”就将进入 `navigateTo` 的 `fail` 函数中。

为什么会出现这样的情况？

我们在 4.11.2 小节中介绍 `redirectTo` 和 `navigateTo` 时，提到过这两个方法只能用于不带 `tab` 选项卡的页面。此时要跳转的 `post` 页面已经被设置成了带选项卡的页面，所以无论使用 `redirectTo` 还是 `navigateTo` 都不能成功跳转，必须使用 4.11.2 小节中我们提到的另外一个导航方法（`wx.switchTab` 方法），才能成功跳转到带有 `tab` 选项卡的页面。

修改 `welcome.js` 页面的 `onTapJump` 方法。

代码清单 10-2 使用 `wx.switchTab` 方法导航

`welcome.js`

```
onTapJump: function (event) {
  wx.switchTab({
    url: "../post/post",
    success: function () {
      console.log("jump success")
    },
    fail: function () {
      console.log("jump failed")
    },
    complete: function () {
      console.log("jump complete")
    }
  });
}
```

以上代码仅仅是将原先所调用的 `wx.navigateTo` 修改成了 `wx.switchTab`。保存并运行代码，此时再次点击 `welcome` 页面的“开启小程序之旅”，可以成功打开 `post` 页面。此时的 `post` 页面底部出现了一个 `tab` 选项卡，如图 10-3 所示。

可以通过点击【文字】和【光影】进行文章和电影页面的切换。

需要特别注意的是, `wx.redirectTo` 和 `wx.navigateTo` 无法跳转到带有 tab 选项卡的页面; 同理, 使用 `wx.switchTab` 也无法跳转到不带 tab 选项卡的页面。它们各司其职, 不能滥用。

开发者还可以尝试一下 tab 选项卡在页面上部的布局, 将 `app.json` 中 `tabBar` 配置下的 `position` 由 “bottom” 修改为 “top”, tab 选项卡将出现在页面上部, 如图 10-4 所示。

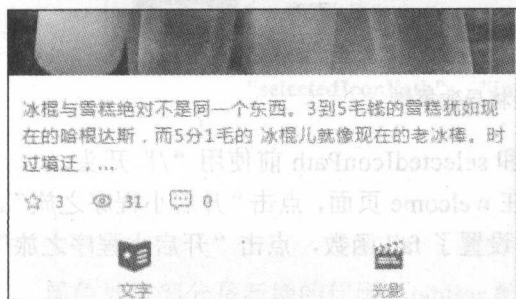


图 10-3 tab 选项卡

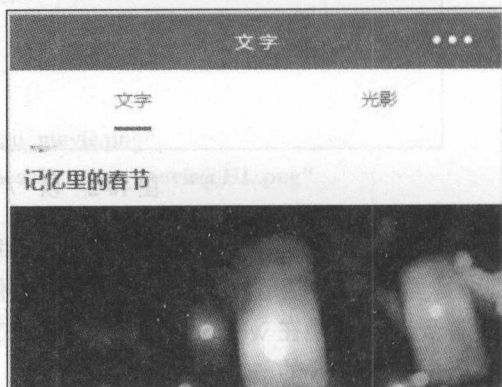


图 10-4 tab 选项卡在上部的布局

可以看到在上部的 tab 选项卡是不包含 tab 图标的, 即使你设置了 tab 的图标也不会出现。

10.2 电影页面介绍

电影模块部分总共有以下几个展示模块:

- 电影首页展示正在热映、即将上映和豆瓣 top250 三种类型的电影。
- 每种电影只展示最前面 3 部。
- 每种电影有一个 “更多” 按钮, 点击将打开一个新页面, 展示该类型下所有的电影。
- 支持电影搜索功能。
- 点击任意一部电影都将打开电影详情页面。

开发者可以参考本书彩页中的设计图, 直观且详细地了解各个功能模块。

所有电影数据均来自于豆瓣电影开放 API, 以下是豆瓣电影 API 文档的地址:

https://developers.douban.com/wiki/?title=movie_v2

对于电影页面的编写, 我们将大量使用 `template` 模板, 甚至是使用多层次嵌套模板。

图 10-5 解释了电影模块中所有页面及模板的结构关系。开发者不需要现在马上看明白上面的结构关系图, 只需要在后续章节中时时回顾一下此图即可。箭头中没有标注包含数量的表示只包含 1 个模板。

我们大概解释一下以上关系调用图: 电影功能部分总共有 3 个页面, 分别是电影首页、更多电影和电影详情页面以及 1 个电影搜索模块 (电影搜索不是单独的一个页面, 位于电影首页)。

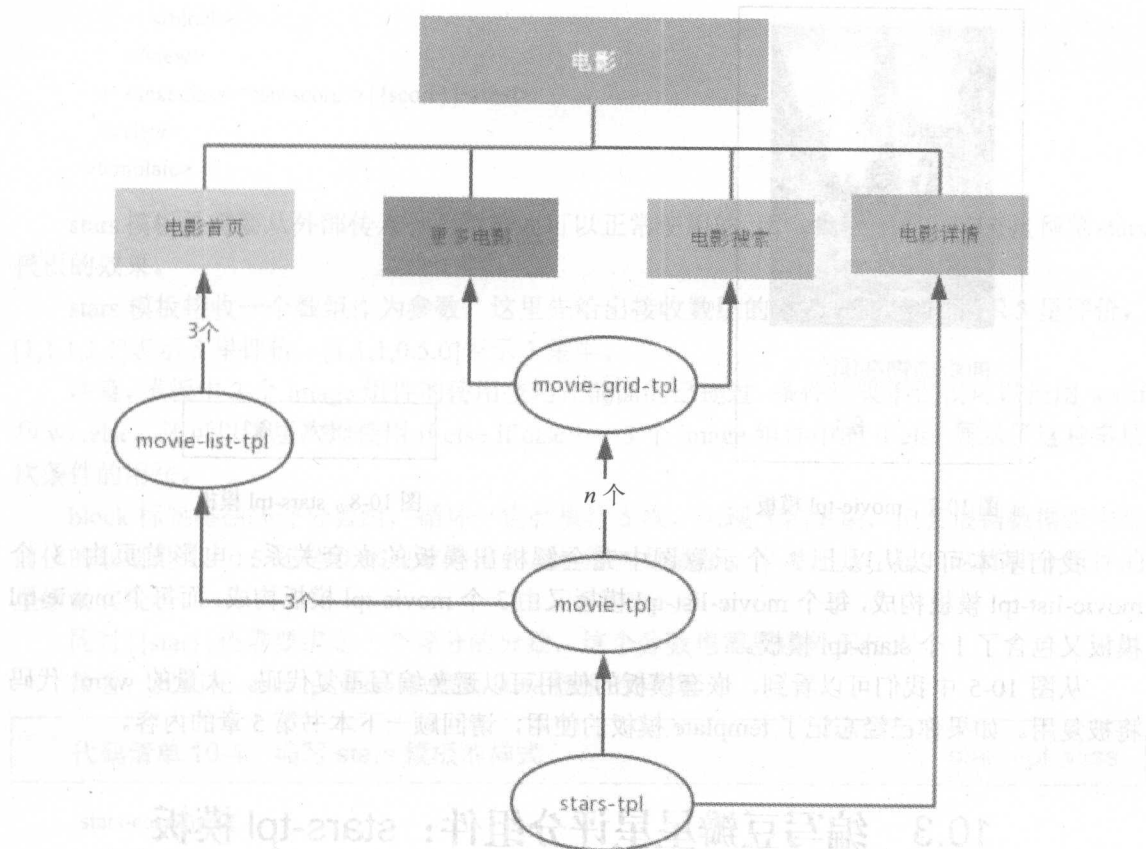


图 10-5 电影页面及模板嵌套关系图

以电影首页为例：电影首页由 3 个 movie-list-tpl 模板构成，每个 movie-list-tpl 模板由 n 个 movie-tpl 模板构成，而每个 movie-tpl 模板又包含 1 个 stars-tpl 模板。

图 10-6~图 10-8 是以上几个模板的示意图。



图 10-6 movie-list-tpl 模板



图 10-7 movie-tpl 模板

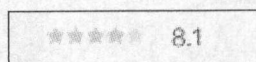


图 10-8 stars-tpl 模板

我们基本可以从以上 3 个示意图中完全解析出模板的嵌套关系：电影首页由 3 个 movie-list-tpl 模板构成，每个 movie-list-tpl 模板又由 3 个 movie-tpl 模板构成，而每个 movie-tpl 模板又包含了 1 个 stars-tpl 模板。

从图 10-5 中我们可以看到，嵌套模板的使用可以避免编写重复代码，大量的 wxml 代码将被复用。如果你已经忘记了 template 模板的使用，请回顾一下本书第 5 章的内容。

10.3 编写豆瓣星星评分组件：stars-tpl 模板

当点击 tab 选项卡的“光影”选项时将跳转到电影首页。

在 10.3~10.5 这 3 节中，我们将连续编写 3 个模板。对于这些模板，我们只需要大概浏览一下它们的骨架结构，对于 {{}} 中所绑定的数据，无须太过关心。当这 3 个模板被编写完成并在电影首页中传入数据调用时自然会明白每个 {{}} 所绑定的数据意义。

在 10.2 节中，我们分析了电影首页的模板构成。下面我们首先编写电影首页所需要的几个模板，这里从最小的模板（stars-tpl 模板）开始编写。

在 movie 目录下新建一个 stars 文件夹，并在该文件夹下新建两个文件：stars-tpl.wxml 和 stars-tpl.wxss。随后在 stars-tpl.wxml 中新增以下代码：

代码清单 10-3 编写 stars 模板的骨架

stars-tpl.wxml

```
<template name="starsTpl">
  <view class="stars-container">
    <view class="stars">
      <block wx:for="{{stars}}" wx:for-item="i">
        <image wx:if="{{i===1}}" src="/images/icon/wx_app_star.png"></image>
        <image wx:elif="{{i===0.5}}" src="/images/icon/wx_app_star@none.png"></image>
        <image wx:else="{{i===0}}" src="/images/icon/wx_app_star@half.png"></image>
```



```

    </block>
  </view>
  <text class="star-score">{{score}}</text>
</view>
</template>

```

stars 模板是需要从外部传入一些数据才可以正常使用的,所以我们现在无法直接预览 stars 模板的效果。

stars 模板接收一个数组作为参数,这里先给出接收数组的形式:[1,1,1,0,0]表示 3 星评价,[1,1,1,1,1]表示 5 星评价,[1,1,1,0.5,0]表示 3 星半。

注意,模板中 3 个 image 组件的使用技巧在前面的已提过,条件渲染不仅仅可以使用 wx:if 和 wx:else,还可以多层次地使用 if else if else……3 个 image 组件中的 if else 展示了这种多层次条件的用法。

block 标签将循环评分数组,循环一定会执行 5 次,出现 5 颗星星,但会根据数据组中当前位的取值是 1、0.5 还是 0 来决定当前的星星图片是满星、半星还是空星。这样评分组件的星级就实现了。

同时 {{star}} 还需要绑定一个评分的分数,这个分数也需要由外部传入。

接着,我们在 stars-tpl.wxss 中编写 stars 模板的样式。

代码清单 10-4 编写 stars 模板的样式

stars-tpl.wxss

```

.stars-container {
  display: flex;
  flex-direction: row;
}

.stars {
  display: flex;
  flex-direction: row;
  height: 17rpx;
  margin-right: 24rpx;
  margin-top: 6rpx;
}

.stars image {
  padding-left: 3rpx;
  height: 17rpx;
  width: 17rpx;
}

.star-score {
  color: #4A6141;
}

```

以上就是 stars-tpl 模板的全部代码，我们暂且放下，接着编写其他模板的代码，最后再将这些模板组装起来。

10.4 编写 movie-tpl 模板

接着我们编写电影首页中需要用到的另外一个模板：movie-tpl。模板的示意图请参考图 10-7。

在 pages 的 movie 目录下新建一个 single-movie 目录，用来存放 movie-tpl 模板。在 /pages/movie/single-movie 目录下新建两个文件：movie-tpl.wxml 和 movie-tpl.wxss 文件。

首先在 movie-tpl.wxml 中编写 movie-tpl 模板的骨架代码：

代码清单 10-5 movie-tpl 模板的骨架代码

movie-tpl.wxml

```
<import src="../../stars/stars-tpl.wxml" />
<template name="movieTpl">
  <view class="movie-container" catchtap="onMovieTap"
    data-movie-id="{{movieId}}">
    <image class="movie-img" src="{{coverageUrl}}"></image>
    <text class="movie-title">{{title}}</text>
    <template is="starsTpl" data="{{{stars:stars, score: average}}}" />
  </view>
</template>
```

在以上代码的顶部我们使用 import 引入了在 10.3 节中定义 stars-tpl 模板。Movie-tpl 模板由一张电影海报、电影标题以及 stars-tpl 模板构成。

对于整个 movie-tpl 模板，我们在其容器上注册了一个事件 onMovieTap，并绑定了当前电影的 id 号——movieId。

需要注意的是，stars-Tpl 模板的 data 属性，该属性将 stars 数组和 score 评分传入 stars-Tpl 中（请开发者回顾一下 10.3 节中的 stars-Tpl 定义）。

那么 movie-tpl 模板中的 stars 数组和 score 评分又是从哪里来的呢？答案依然是从外部传入的。在 10.5 节中编写 movie-list-tpl 时将看到如何传入这两个参数。

接着编写 movie-tpl 模板的样式。

代码清单 10-6 编写 movie-tpl 模板的样式

movie-tpl.wxss

```
@import "../../stars/stars-tpl.wxss";

.movie-container {
  display: flex;
  flex-direction: column;
  padding: 0 22rpx;
```



```

}

.movie-img {
  width: 200rpx;
  height: 270rpx;
  padding-bottom: 20rpx;
}

```

```

.movie-title{
  margin-bottom: 16rpx;
  font-size: 24rpx;
}

```

注意，在样式的顶部我们依然需要引入 stars-tpl 模板所使用的样式。

10.5 编写 movie-list-tpl 模板

直接被电影首页调用的模板是 movie-list-tpl 模板，movie-list-tpl 模板中引用了 movie-tpl 模板，而 movie-tpl 模板中又引用了 stars-tpl 模板。这就是我们所说的 3 层模板嵌套关系。

首先在 pages/movie 目录下新建 movie-list 目录，接着在 movie-list 目录下新建两个文件：movie-list-tpl.wxml 和 movie-list-tpl.wxss 文件。

下面编写 movie-list-tpl 模板的骨架。在 movie-list-tpl.wxml 中添加以下代码：

代码清单 10-7 movie-list-tpl 模板的骨架

movie-list-tpl.wxml

```

<import src="../../single-movie/movie-tpl.wxml" />
<template name="movieListTpl">
  <view class="movie-list-container">
    <view class="inner-container">
      <view class="movie-head">
        <text class="slogan">{{categoryTitle}}</text>
        <view catchtap="onMoreTap" class="more" data-category="{{categoryTitle}}">
          <text class="more-text">更多</text>
          <image class="more-img" src="/images/icon/wx_app_arrow_right.png"></image>
        </view>
      </view>
      <view class="movies-container">
        <block wx:for="{{movies}}" wx:for-item="movie">
          <template is="movieTpl" data="{{...movie}}"/>
        </block>
      </view>
    </view>
  </view>
</template>

```

```
</view>
</template>
```

同样，我们在代码的开头部分使用 `import` 引入 `movie-tpl` 模板，并在 `block` 标签中使用 `movie-tpl` 这个模板。注意 `template` 标签中的 `data` 属性，我们在这里将 `movie-tpl` 模板所需要的数据传进去。

接着编写 `movie-list-tpl` 模板的样式。在 `movie-list-tpl.wxss` 文件中编写以下代码：

代码清单 10-8 编写 `movie-list-tpl` 样式

`movie-list-tpl.wxss`

```
@import "../single-movie/movie-tpl.wxss";
```

```
.movie-list-container {
  background-color: #fff;
  display: flex;
  flex-direction: column;
}
```

```
.inner-container{
  margin: 0 auto 20px;
}
```

```
.movie-head {
  padding: 30px 20px 22px;
}
```

```
.slogan {
  font-size: 24px;
}
```

```
.more {
  float: right;
}
```

```
.more-text {
  vertical-align: middle;
  margin-right: 10px;
  color: #4A6141;
}
```

```
.more-img {
  width: 9px;
  height: 16px;
  vertical-align: middle;
```




```
}  
  
.movies-container{  
  display: flex;  
  flex-direction: row;  
}  
}
```

注意在代码的开头部分使用 `import` 引入 `movie-list-tpl` 模板所使用的 `movie-tpl` 模板的样式文件 `movie-tpl.wxss`。

10.6 电影首页的骨架与样式

在我们编写好 3 个 `template` 组件后, 电影首页的骨架编写将变得非常简单, 只需要在电影首页中引用 `movie-list-tpl` 模板即可。

在 `movie.wxml` 页面中添加以下代码:

代码清单 10-9 编写电影首页骨架代码

movie.wxml

```
<import src="movie-list/movie-list-tpl.wxml" />  
  
<view class="container">  
  <view class="movies-template">  
    <template is="movieListTpl" data="{{...inTheaters}}" />  
  </view>  
  
  <view class="movies-template">  
    <template is="movieListTpl" data="{{...comingSoon}}" />  
  </view>  
  
  <view class="movies-template">  
    <template is="movieListTpl" data="{{...top250}}" />  
  </view>  
</view>
```

注意, 在代码开头部分使用 `import` 引入了 `movie-list-tpl` 模板。同时, 在本段代码中使用了 3 次 `movie-list-tpl` 模板, 分别代表 `inTheaters` 正在热映、`comingSoon` 即将上映和 `top250` 经典电影 3 种类型的电影。

接着编写电影首页的样式。在 `movie.wxss` 文件中添加以下代码:

代码清单 10-10 编写电影首页的样式

movie.wxss

```
@import "movie-list/movie-list-tpl.wxss";
```



```
.container {  
  background-color: #f2f2f2;  
}  
.movies-template {  
  margin-bottom: 30px;  
}
```

10.7 豆瓣电影 API 分析

在编写电影首页的 js 调用豆瓣 API 之前，我们首先应当对豆瓣电影 API 有一个直观的认识和了解。所谓开放 API，是指某些公司、企业将自己公司所持有的数据、用户数据选择性地开放给开发者调用，让开发者可以使用数据并围绕这些数据构建自己的应用，从而帮助公司、企业完善其平台和生态。

目前，绝大多数的开放 API 都属于 RESTful 风格的 API，比如豆瓣 API、github 开发者 API 等。豆瓣 API（V2 版）就属于这类 API，其 API 权限有 3 种：

- 公开
- 高级
- 商务

所有开发者无须申请即可调用公开权限的 API，但只有部分 API 是公开权限的，一些高级接口无法调用，且公开 API 具有 40 次/分钟的访问限制。如果开发者在调试代码时遇到豆瓣返回错误信息，就有可能是因为已经超过允许访问的次数。这个时候唯一的办法就是稍等片刻。

高级权限和商务权限需要开发者向豆瓣发邮件申请。

本书中所调用的豆瓣 API 均为公开 API，所以要注意访问频率限制的问题。在 Orange Can 项目中总共使用了以下几个豆瓣电影 API：

- 获取正在热映的电影。
- 获取即将上映的电影。
- 获取豆瓣 top250 电影。
- 电影搜索。
- 获取电影条目信息（电影详情数据）。

以获取豆瓣 top250 的 API 地址为例，一个完整的获取豆瓣 top250 电影的 API 地址为：

<https://api.douban.com/v2/movie/top250?start=0&count=15>

在以上 API 中，我们向豆瓣请求 top250 电影中的前 15 条电影信息。start=0 和 count=15 是查询参数，可根据需求自行调整，但要注意每次加载最多只能获取 20 条数据，count 超过 20 是没有意义的。



开发者可直接在浏览器中输入以上 URL 地址，豆瓣将返回我们需要的数据，返回数据的类型是 JSON 类型，如图 10-9 所示。

```

{
  "genre": [
    "剧情",
    "爱情"
  ],
  "title": "阿甘正传",
  "casts": [
    {
      "alt": "https://movie.douban.com/celebrity/1054450/",
      "avatars": {
        "small": "https://img3.doubanio.com/img/celebrity/small/551.jpg",
        "large": "https://img3.doubanio.com/img/celebrity/large/551.jpg",
        "medium": "https://img3.doubanio.com/img/celebrity/medium/551.jpg"
      },
      "name": "汤姆·汉克斯",
      "id": "1054450"
    },
    {
      "alt": "https://movie.douban.com/celebrity/1002676/",
      "avatars": {
        "small": "https://img1.doubanio.com/img/celebrity/small/51737.jpg",
        "large": "https://img1.doubanio.com/img/celebrity/large/51737.jpg",
        "medium": "https://img1.doubanio.com/img/celebrity/medium/51737.jpg"
      },
      "name": "罗宾·怀特"
    }
  ]
}

```

图 10-9 豆瓣电影返回数据（部分）

关于其他几个 API 接口的使用方法，我们将放在后续实例编码中讲解。

10.8 电影首页的 js 编写

电影首页 js 的编码工作主要做以下 3 件事情：

- 调用豆瓣 API 获取电影数据。
- 处理豆瓣电影数据。
- 绑定豆瓣电影数据。

首先在 app.js 中的 globalData 中加入一个全局变量 doubanBase，用来记录豆瓣 API 的基地址。因为所有豆瓣的 API 访问都需要这个基地址，所以我们将它单独放在全局变量中，并在其他需要的地方获取这个全局变量。

代码清单 10-11 豆瓣 API 的基地址

app.js

```

globalData: {
  g_isPlayingMusic: false,
  g_currentMusicPostId: null,
  doubanBase: "https://api.douban.com"
}

```

黑色加粗部分为新增代码。

接着编写 movie.js 中的代码。首先在 movie.js 中新增 Page 方法，并在 Page 方法的参数中定义 data 初始化数据及页面 onLoad 函数。

代码清单 10-12 编写 movie 页面的 Page 方法

movie.js

```
var app = getApp();

Page({
  data: {
    inTheaters: {},
    comingSoon: {},
    top250: {}
  },

  onLoad: function (event) {
    var inTheatersUrl = app.globalData.doubanBase +
      "/v2/movie/in_theaters" + "?start=0&count=3";
    var comingSoonUrl = app.globalData.doubanBase +
      "/v2/movie/coming_soon" + "?start=0&count=3";
    var top250Url = app.globalData.doubanBase +
      "/v2/movie/top250" + "?start=0&count=3";

    this.getMovieListData(inTheatersUrl, "inTheaters", "正在热映");
    this.getMovieListData(comingSoonUrl, "comingSoon", "即将上映");
    this.getMovieListData(top250Url, "top250", "豆瓣 Top250");
  }
})
```

在 onLoad 函数中，主要拼接了 3 个不同的豆瓣 API，分别用来获取正在热映、即将上映和豆瓣 top250 电影的数据。

这 3 个 API 唯一的区别在于 URL 路径中的一小段路径/v2/movie/key 中的 key 不同：“正在热映”的 key 是 inTheaters，“即将上映”的 key 是 comingSoon，“top250”的 key 是 top250。

注意，我们使用“?start=0&count=3”指定仅获取每种类型电影的前 3 条数据，因为在我们的业务需求中只需要每种类型的前 3 条数据。

在 data 属性中，预先定义了 3 个对象：inTheaters、comingSoon 和 top250。这 3 个对象将用于数据绑定，开发者可回顾一下代码清单 10-9。在代码清单 10-9 中，分别将这 3 个对象绑定到了 template 模板上。

在代码清单 10-12 的末尾部分，调用了 3 次 this.getMovieListData 方法。这个方法的作用就是根据传入的 url 获取和处理数据。在 10.9 节中，我们将编写这个方法。



10.9 wx.request 发送 http/https 请求

目前我们还没有编写 `getMovieListData` 方法，现在在 `movie.js` 的页面中添加这个方法。

代码清单 10-13 编写 `getMovieListData` 方法

movie.js

```
getMovieListData: function (url, settedKey, categoryTitle) {
  var that = this;
  wx.request({
    url: url,
    method: 'GET',
    header: {
      "content-type": "json"
    },
    success: function (res) {
      that.processDoubanData(res.data, settedKey, categoryTitle)
    },
    fail: function (error) {
      // fail
      console.log(error)
    }
  })
}
```

`getMovieListData` 接受 3 个参数：`url`、`settedKey` 和 `categoryTitle`。`url` 被用来访问并获取豆瓣电影数据，`settedKey` 被用来作为不同类型电影数据绑定的 `key`，`categoryTitle` 被用来作为电影的分类标题，最终将被显示在 `movie-list-tpl` 的标题上（参考代码清单 10-7）。

`getMovieListData` 的核心代码是 `wx.request` 方法的使用，是小程序提供的发送 `http` 和 `https` 请求的方法，类似于传统 `JavaScript` 编程中所使用的 `ajax` 方法。

`wx.request(object)` 方法用于发送 `http/https` 请求，并接受服务器返回的请求结果。`Object` 参数包含以下几个属性用于配置请求参数：

- `url` `String` 类型，开发者服务器接口地址。
- `data` `Object` 或者 `String` 类型，请求的参数。
- `header` `Object` 类型，设置请求的 `header`。注意，在 `header` 中不能设置 `referer`，因为在小程序中 `referer` 是一个固定格式的值，格式固定为 `https://servicewechat.com/{appid}/{version}/page-frame.html`，其中 `{appid}` 为小程序的 `appid`、`{version}` 为小程序的版本号，版本号为 0 表示开发版。
- `method` `String` 类型，默认为 `GET`，有效值为 `OPTIONS`、`GET`、`HEAD`、`POST`、`PUT`、`DELETE`、`TRACE`、`CONNECT`。注意，`method` 的取值必须大写。

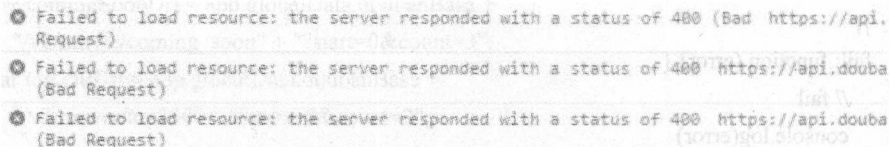


- `dataType` String 类型，默认为 `json`。如果设置了 `dataType` 为 `json`，就会尝试对响应的数据做一次 `JSON.parse`。
- `success` Function 类型，收到开发者服务成功返回的回调函数。函数会被传入一个 `res` 参数，开发者可通过 `res.data` 来获取服务器返回的内容。
- `fail` Function 类型，接口调用失败的回调函数。
- `Complete` Function 类型，接口调用结束的回调函数（调用成功、失败都会执行）。

开发者可能注意到，在代码清单 10-13 中，我们在设置 `header` 的 `content-type` 时给出了一个奇怪的参数值：`json`。既没有给出常见的 `application/json`，也没有直接省略掉 `content-type` 这个选项（官方文档中明确指出 `content-type` 的默认值是“`application/json`”）。原因是以下几个选项都无法正确调用豆瓣 API：

- 不设置 `content-type`
- `content-type:'application/json'`
- `content-type:''`

在目前的 130400 版本中，如果将 `content-type` 设置为以上几个取值，那么豆瓣 API 将返回如图 10-10 所示的错误信息。



```

❌ Failed to load resource: the server responded with a status of 400 (Bad Request)
❌ Failed to load resource: the server responded with a status of 400 (Bad Request)
❌ Failed to load resource: the server responded with a status of 400 (Bad Request)

```

图 10-10 `content-type` 设置导致的错误提示

经过反复调试发现，`content-type` 可以被指定为任意不为空的字符串：

- `content-type:'json'`
- `content-type:'xxxxxx'`
- `content-type:'appli'`

只要 `content-type` 不为空或者是 `application/json`，就可以正确调用豆瓣 API 并获取到豆瓣的返回数据。

为了让 `content-type` 看起来合理一些，在 `Orange Can` 项目中一律使用 `content-type:'json'` 的形式作为 `content-type` 的取值。

对于 `wx.request`，还有以下几个注意事项：

- `url` 中不能有端口。
- `wx.request` 的默认超时时间和最大超时时间都是 60s。
- `wx.request` 的最大并发数是 5。

10.10 设置 wx.request 的超时时间

很多开发者可能想知道如何在小程序中设置超时时间，答案是在 `app.json` 文件中配置这个超时时间。

在 `app.json` 中除了我们之前讲到的 `pages`、`window`、`tabBar` 等常用的配置项外，还有一个 `networkTimeout` 配置项。`networkTimeout` 配置项用来配置各类网络请求的超时时间：

- `request` `wx.request` 的超时时间，单位为毫秒，默认值为 60000。
- `connectSocket` `wx.connectSocket` 的超时时间，单位为毫秒，默认值为 60000。
- `uploadFile` `wx.uploadFile` 的超时时间，单位为毫秒，默认值为 60000。
- `downloadFile` `wx.downloadFile` 的超时时间，单位为毫秒，默认值为 60000。

我们可以在 `app.json` 中添加以下代码来设置各类请求的超时时间：

代码清单 10-14 设置请求的超时时间

`app.json`

```
"networkTimeout": {
  "request": 20000,
  "connectSocket": 20000,
  "uploadFile": 20000,
  "downloadFile": 20000
}
```

当然，如果目前位置只使用 `wx.request` 请求，只设置 `request` 这个配置项也是可以的。如果服务器在 20 秒内没有响应，那么 `wx.request` 将进入 `fail` 函数。

10.11 处理返回的电影数据

在代码清单 10-13 中，我们使用 `wx.request` 来获取豆瓣 API 所提供的电影数据。需要注意的是，`wx.request` 是一个异步方法，且小程序只提供了异步发送 `http/https` 请求的方法，没有同步版本。所以，不要尝试使用以下方法来获取 `wx.request` 的返回值：

```
var result = wx.request(object)
```

对于同步的方法，可以使用以上方式来获取返回结果；但对于异步方法，只能在异步方法所提供的回调函数中进行返回结果的处理。比如在 `wx.request` 中，就只能在 `wx.request` 所提供的 `success` 回调函数中处理豆瓣的返回结果。

回顾一下代码清单 10-13，在 `success` 回调函数中，我们使用 `res.data` 来获取豆瓣 API 的返回结果，接着调用了一个 `processDoubanData` 方法来处理豆瓣 API 的返回数据，从豆瓣加载回



来的数据格式如图 10-11 所示。开发者可自行使用 Postman 或者 Fiddler 来调用豆瓣 API 查看返回的数据格式。

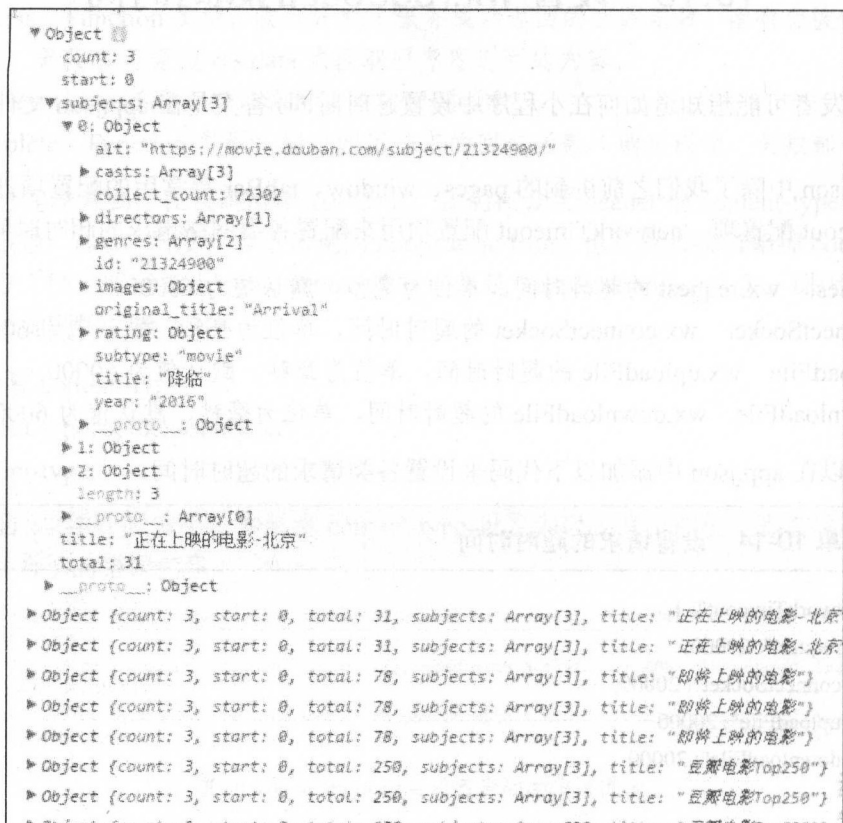


图 10-11 豆瓣返回的电影数据

目前，我们还没有实现 processDoubanData，下面来编写这个函数。

代码清单 10-15 处理豆瓣电影数据

movie.js

```

processDoubanData: function (moviesDouban, settedKey,
categoryTitle) {
  var movies = [];
  // for 中的代码将所有豆瓣电影数据转化成我们需要的格式
  for (var idx in moviesDouban.subjects) {
    var subject = moviesDouban.subjects[idx];
    var title = subject.title;
    if (title.length >= 6) {
      //电影标题只取前 6 个字符
      title = title.substring(0, 6) + "...";
    }
    var temp = {
      stars: util.convertToStarsArray(subject.rating.stars),

```




```

        title: title,
        average: subject.rating.average,
        coverageUrl: subject.images.large,
        movieId: subject.id
    }
    movies.push(temp)
}
var readyData = {};
readyData[settedKey] = {
    categoryTitle: categoryTitle,
    movies: movies
};
this.setData(readyData);
}

```

这个函数接收 3 个参数，第一个参数 `moviesDouban`，就是从豆瓣获取来的电影数据，`processDoubanData` 的主要功能就是处理 `moviesDouban`；第二个、第三个参数在之前已经解释过，这里只是简单地传入到 `processDoubanData` 中，在最后才绑定到 `setData` 方法中。

`for` 循环将遍历所有豆瓣电影的数据，并将豆瓣的数据格式转化成我们需要的命名格式，`temp` 就是一个临时保存转化后电影数据的变量。

最后将所有豆瓣电影数据处理完后调用 `js` 的 `Array.push` 方法将 `temp` 加入 `movies` 数组中，最终形成我们需要的电影数据数组。

在 `for` 循环中处理豆瓣电影数据时，调用了一个 `util.convertToStarsArray` 方法将豆瓣的评分格式转化成我们需要的数组格式——形如 `[1,1,0,0,0]` 的形式（开发者可回顾一下 10.3 节）。豆瓣 API 返回的关于评分的数据格式为 50（表示 5 星）、35（表示 3 星半）、00（0 星或者还没有星级），`util.convertToStarsArray` 的任务就是将这些 50、35、00 等星级的简写形式转化成 `[1,1,1,1,1]`、`[1,1,1,0.5,0]` 和 `[0,0,0,0,0]` 等需要的数组形式，以方便 `stars-tpl` 模板使用。

在 `util.js` 中加入一个 `convertToStarsArray` 函数。

代码清单 10-16 评分的格式转化函数

util.js

//将 50、35、00 等形式转化成[1,1,1,1,1]的形式

```

function convertToStarsArray(stars) {
    var num = stars / 10;
    var array = [];
    for (var i = 1; i <= 5; i++) {
        if (i <= num) {
            array.push(1);
        }
        else {
            if ((i - num) === 0.5) {
                array.push(0.5)
            }
        }
    }
}

```

```

else{
  array.push(0);
}
}
}
return array;
}

```

记住，util 是一个模块，添加完方法后需要使用 `module.exports` 输出。

代码清单 10-17 输出 `convertToStarsArray` 函数

util.js

```

module.exports = {
  getDiffTime: getDiffTime,
  convertToStarsArray: convertToStarsArray
}

```

黑色加粗部分为新增代码。

随后，要想在 `movie.js` 中引用 `util`，还需要在 `movie.js` 顶部引用 `util` 模块。

代码清单 10-18 引用 `util` 模块

movie.js

```

var util = require('../util/util.js')

```

完成以上代码的补充后，我们才能够在 `movie.js` 中调用 `util.convertToStarsArray` (`subject.rating.stars`)方法。

10.12 绑定处理后的电影数据

在代码清单 10-15 的最后我们使用了以下代码进行数据绑定：

代码清单 10-19

```

var readyData = {};
readyData[settedKey] = {
  categoryTitle: categoryTitle,
  movies: movies
}
this.setData(readyData);

```

这是一种动态设置数据绑定 `key` 的方法。由于我们并不知道当前处理的数据是哪一种电影类型（`inTheaters`、`comingSoon`、`top250`），因此将当前所处理的电影数据类型通过 `settedKey` 一路传递到 `processDoubanData` 方法中，并通过 `readyData[settedKey]` 生成一个包含 `settedKey` 的 JavaScript 对象。



假设当前处理的数据是 `inTheaters` 类型，那么以上代码在最终调用 `this.setData(readyData)` 时相当于以下形式：

代码清单 10-20

```
this.setData({
  inTheaters:{
    categoryTitle:'正在热映',
    movies: movies
  }
})
```

如果当前处理的数据是 `comingSoon` 类型，那么以上代码在最终调用 `this.setData` 时相当于以下形式：

代码清单 10-21

```
this.setData({
  comingSoon:{
    categoryTitle:'即将上映',
    movies: movies
  }
})
```

这样的写法实际上考验的是开发者对 JavaScript 动态属性的理解。此时，保存并运行代码，电影首页显示如图 10-12 所示的样式。



图 10-12 电影首页的最终运行效果

如果数据没有正确加载，请参考下节的内容。

10.13 http 和 https 在小程序中的使用说明

为了保证数据的安全性，小程序中强制要求使用 https，且所访问的 https 地址必须在小程序的后台账号中被加入到可信域名中。图 10-13 是小程序开发账号的 https 可信域名配置的示意图。

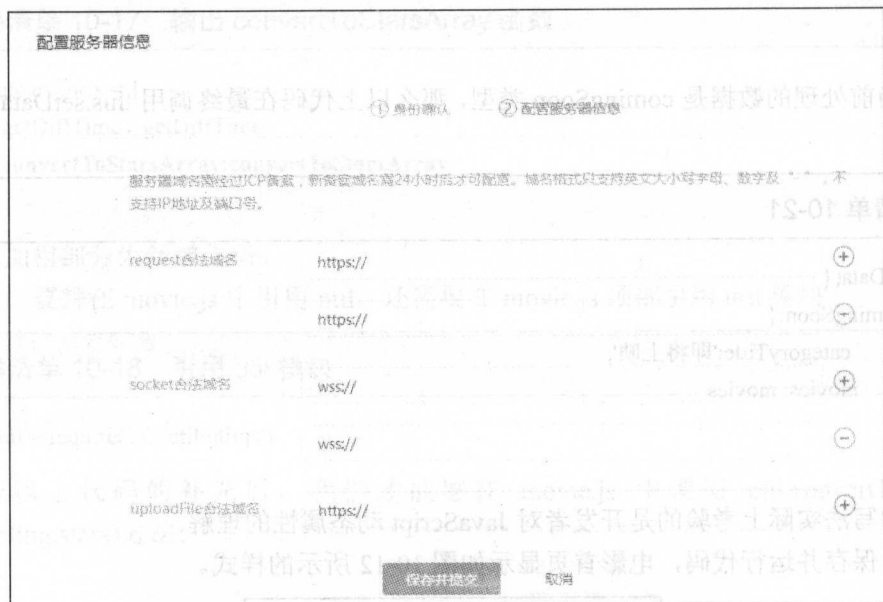


图 10-13 可信域名的配置

该配置项位于小程序账号的“设置”→“开发设置”选项中。理论上，小程序是不允许使用 http 请求来获取数据的，也不允许访问未在可信域名列表中配置的 https 地址。

以上限制的前提条件是：在真机上。

如果是在客户端的开发工具中，有以下几种方法可以不需要遵守以上两个规则。

(1) 第一种方法

如果在创建小程序项目时选取的是无 appid（必须拥有小程序账号才可以获得），那么开发工具不会限制你访问 https，更不会比对可信域名列表。这种方法的缺点就是：当你偶尔想真机预览一下小程序时，还需要重新新建项目并填入 appid。

(2) 第二种方法

如果在创建小程序项目时填入了 appid，那么默认情况下小程序将强制你使用 https 且会将你所访问的 https 地址与可信域名列表做比对。如果你试图在一个配置了 appid 的小程序项目客户端开发工具中访问 http 地址，那么开发工具会报错，如图 10-14 所示。



Thu Jan 26 2017 16:15:48 GMT+0800 (中国标准时间) 合法域名校验出错

http://api.douban.com/v2/movie/coming_soon?start=0&count=3 不在以下合法域名列表中, 请参考文档: https://mp.weixin.qq.com/debug/wxadoc/dev/api/network-request.html

图 10-14 访问的地址不在可信域名列表中

无论你使用 http 或者使用不在可信域名列表里的 https 地址都将提示图 10-14 所示的错误(可信域名列表不能够设置 http, 所以试图访问 http 地址一样会提示不在可信域名列表中)。

解决的办法是, 在可信域名列表中配置你要访问的 https 域名。

(3) 第三种方法

小程序在开发工具侧边栏的“项目”选项卡中提供了“开发环境不校验请求域名以及 TLS 版本”的选项, 勾选这个选项你可以随意在开发工具中使用 http 或者未加入 https 可信列表的 https 地址。

以下是一些建议:

- 如果你没有 appid, 那么只能选择无 appid。
- 如果你有 appid, 那么建议在新建项目时填写这个 appid, 并在开发期间勾选“开发环境不校验请求域名以及 TLS 版本”, 待正式发布或者需要真机预览时再去小程序账号中配置你所使用的 https 地址。如果项目中没有使用网络请求, 那么真机预览时也不需要配置 https 的可信域名列表。
- 如果你在 Orange Can 项目中遇到无法访问的问题, 可以参考上面所描述的内容。

10.14 跳转到更多电影页面

之前, 我们完成了电影首页的编写。电影首页总共展现了 3 种类型的电影, 共 9 部。如果我们需要查看每种类型的全部电影, 就需要编写“更多电影”这个页面。“更多电影”是通过点击电影首页的“更多”按钮来打开新页面的, 如图 10-15 所示。



图 10-15 右上角的“更多”按钮将打开“更多电影”页面

首先新建 more-movie 页面。在 app.json 文件的 pages 数组下新增 more-movie 页面的路径，如下代码所示：

代码清单 10-22 添加 more-movie 页面的页面路径

app.json

```
"pages": [
  "pages/welcome/welcome",
  "pages/post/post",
  "pages/post/post-detail/post-detail",
  "pages/post/post-comment/post-comment",
  "pages/movie/movie",
  "pages/movie/more-movie/more-movie"
]
```

黑色加粗部分代码是新增的页面路径。添加以上代码后保存项目，将自动生成 more-movie 页面的 4 个页面文件。

我们首先实现跳转到 more-movie 页面的代码。通过图 10-15 可以看到，“更多”按钮是位于 movie-list-tpl 模板中的。我们回顾一下 movie-list-tpl.wxml 中的代码。

在 movie-list-tpl.wxml 中的<view class="more">标签上，我们已经注册了一个 onMoreTap 事件，且在这个标签上也已经绑定了一个 data 数据：data-category="{{categoryTitle}}"。

实现跳转到 more-movie 页面的关键就在于实现这个 onMoreTap 函数，并在 onMoreTap 函数中使用 wx.navigateTo 方法跳转到 more-movie 页面中去。同时，我们还必须将 categoryTitle 这个变量同时传递到 more-movie 页面里去，从而使 more-movie 页面能够准确地加载“更多”的电影数据。

在 movie.js 中新增以下代码，以实现 onMoreTap 方法。开发者可能会觉得奇怪，“更多”按钮是位于 movie-list-tpl 模板中的，但响应这个按钮的 js 方法却要写在另外一个页面的 js 文件（movie.js）中。

这确实是非常奇怪也难以理解的。关于这个问题，我们已经在前面的内容中讲解过：小程序只实现了模板化，而没有实现组件化，模板是不具备运行 JavaScript 代码能力的。所以，我们只能将模板的业务逻辑编写在其他页面的 js 文件中，再将处理好的业务数据传递到模板中。

代码清单 10-23 编写 onMoreTap 方法

movie.js

```
onMoreTap: function (event) {
  var category = event.currentTarget.dataset.category;
  wx.navigateTo({
    url: "more-movie/more-movie?category=" + category
  })
}
```

代码非常简单，这里就不再做过多得解释了。



加入以上代码后,保存并运行项目,点击电影首页中的“更多”按钮将可以打开 more-movie 页面。

10.15 编写 movie-grid-tpl 模板

如同我们在实现电影首页页面时所做的,我们首先来编写“更多电影”页面所需要的模板: movie-grid-tpl 模板。

图 10-16 展示了 movie-grid-tpl 模板的实际效果图。



图 10-16 movie-grid-tpl 模板效果图

这个模板将像“九宫格”一样放入很多的电影(真实的数据远不止 9 部电影),以展示更多电影的效果。

在/pages/movie 下新建一个 movie-grid 目录,并在目录下新建 movie-grid-tpl.wxml 和 movie-grid-tpl.wxss 文件。

我们首先来编写 movie-grid-tpl 模板的骨架。在 movie-grid-tpl.wxml 文件中加入以下代码:

代码清单 10-24 movie-grid-tpl 骨架

movie-grid-tpl.wxml

```
<import src="../../single-movie/movie-tpl.wxml" />
<template name="movieGridTpl">
  <view class="grid-container">
    <block wx:for="{{movies}}" wx:for-item="movie">
      <view class="single-view-container">
        <template is="movieTpl" data="{{...movie}}" />
      </view>
    </block>
  </view>
</template>
```

我们可以看到 movie-grid-tpl 的骨架代码非常简单。为什么这么几句代码就可以实现复杂的“九宫格”效果？

模板再次发挥了巨大的复用作用。很明显 movie-grid-tpl 仅仅起到的是组织布局的作用，而核心的骨架代码已经被封装在了 movie-tpl 模板中。我们在 movie-grid-tpl 中的 block 标签中循环调用 movie-tpl 模板，以实现显示多部电影的目的。

在之前编写的 movie-list-tpl 里，我们已经使用过了 movie-tpl，这里再次使用 movie-tpl 就体现了模板的优势，它避免了我们重复编写 wxml 代码。笔者认为用好模板将大大简化代码，提高代码的可阅读性与可维护性。

接着编写 movie-grid-tpl 模板的样式代码。在 movie-grid-tpl.wxss 文件中加入以下样式代码：

代码清单 10-25 movie-grid-tpl 模板的样式

movie-grid-tpl.wxss

```
@import "../../single-movie/movie-tpl.wxss";

.single-view-container{
  float:left;
  margin-bottom: 40rpx;
}

.grid-container{
  height: 1300rpx;
  margin:40rpx 0 40rpx 6rpx;
}
```

同样要注意引入 movie-tpl.wxss 文件。



10.16 编写“更多电影”页面

在编写完 more-movie 页面所需要的核心 template 模板后, 我们来编写 more-movie.wxml 文件。令人吃惊的是, 你几乎不需要编写任何代码, 只需要在 more-movie.wxml 页面中引入 movie-grid-tpl 模板即可。

在 more-movie.wxml 页面中加入以下代码:

代码清单 10-26 more-movie 页面的骨架

more-movie.wxml

```
<import src="../../movie-grid/movie-grid-tpl.wxml" />
<template is="movieGridTpl" data="{{movies}}" />
```

接着编写 more-movie 页面的样式。在 more-movie.wxss 文件中加入以下代码:

代码清单 10-27 more-movie 页面的样式

more-movie.wxss

```
@import "../../movie-grid/movie-grid-tpl.wxss";
```

几乎不需要编写代码, 我们就完成了 more-movie 页面的骨架和样式代码的编写工作。

接着我们还需要编写 more-movie 页面的 js 代码。more-movie 页面的 js 代码所需要完成的工作几乎同 movie 电影首页的 js 代码相同。依然遵守着获取数据→处理数据→绑定数据的步骤。

实际上, 获取数据→处理数据→绑定数据的流程几乎是小程序 js 文件编写的通用思路与步骤。在 more-movie 页面中添加以下代码:

代码清单 10-28 编写 more-movie 的 js 代码

more-movie.js

```
var app = getApp()
var util = require('../../util/util.js')
Page({
  data: {
    movies: []
  },
  onLoad: function (options) {
    var category = options.category;
    var dataUrl = "";
    switch (category) {
      case "正在热映":
        dataUrl = app.globalData.doubanBase +
          "/v2/movie/in_theaters";
        break;
```



```

    case "即将上映":
        dataUrl = app.globalData.doubanBase +
            "/v2/movie/coming_soon";
        break;
    case "豆瓣 Top250":
        dataUrl = app.globalData.doubanBase + "/v2/movie/top250";
        break;
    }
    util.http(dataUrl, this.processDoubanData)
},

processDoubanData: function (moviesDouban) {
    var movies = [];
    for (var idx in moviesDouban.subjects) {
        var subject = moviesDouban.subjects[idx];
        var title = subject.title;
        if (title.length >= 6) {
            title = title.substring(0, 6) + "...";
        }
        var temp = {
            stars: util.convertToStarsArray(subject.rating.stars),
            title: title,
            average: subject.rating.average,
            coverageUrl: subject.images.large,
            movieId: subject.id
        }
        movies.push(temp)
    }
    this.setData({
        movies: movies
    });
}
});

```

首先，我们在 data 中设置 movies 绑定变量的初始化值。movies 变量将最终被用作 wxml 的数据绑定变量。

接着，在 onLoad 函数中接收由 movie 电影首页传递过来的 category 分类。根据分类的不同拼接不同数据访问 API 的地址。这里我们没有在 API 的 URL 后面附带 start 和 count 参数，如果不附带这两个参数，那么默认将一次加载 20 条电影数据。

为了避免反复编写 wx.request 的复杂结构，我们在 util 模块中编写了一个 http 方法，作为所有豆瓣 API 调用的公共方法，当我们需要访问豆瓣 API 时，不需要重复调用 wx.request 方法，只需要调用这个封装好的 http 方法即可。util.http 方法接收两个参数：第一个参数是需要访问的 API 地址；第二个参数是一个回调函数，用来处理豆瓣 API 的返回结果。

processDoubanData 方法同 movie 电影首页中的 processDoubanData 类似,都是用来处理豆瓣的返回数据,并在处理完成后将处理的数据进行绑定。由于在 more-movie 页面中我们已经明确了电影类型,因此 processDoubanData 的编写简单了很多,只需要明确处理“正在热映”“即将上映”和“top250”3 种类型中的一种即可。

下面来看看 util.http 方法是如何编写的。在 util.js 文件中加入 http 方法。

代码清单 10-29 编写 util.http 方法

util.js

```
function http(url, callBack) {
  wx.request({
    url: url,
    method: 'GET',
    header: {
      "content-type": "json"
    },
    success: function (res) {
      callBack(res.data);
    },
    fail: function (error) {
      console.log(error)
    }
  })
}
```

将所有调用豆瓣 API 的操作封装成一个函数的好处是,一旦调用操作和流程发生变化,只需要修改这个函数即可。比如,header 的 content-type 支持 application/json 取值,无须在多处修改,只需要修改 util.http 方法即可。

编写完 http 方法后,记得使用 module.exports 方法将 http 方法暴露出去,以供其他页面/模块访问。

代码清单 10-30 暴露 http 方法

util.js

```
module.exports = {
  getDiffTime: getDiffTime,
  convertToStarsArray:convertToStarsArray,
  http:http
}
```

黑色加粗部分为新增代码。

保存并运行项目,再次点击“更多”按钮,页面将从 movie 电影首页跳转到 more-movie (更多电影) 页面,且 more-movie 页面正确地呈现出 20 部电影,如图 10-17 所示。



图 10-17 more-movie 页面的显示效果

10.17 实现页面下拉刷新的“三部曲”

下拉刷新是 APP 上经典的一个动作。本节我们将学习如何在小程序中实现下拉刷新数据的功能。在小程序中，不需要自己实现下拉刷新代码编写。小程序已经为我们准备好了下拉刷新的相关配置和 API。

实现一个页面的下拉刷新操作需要分为 3 步：

- 步骤 01** 在页面的 json 文件中配置 enablePullDownRefresh 选项，打开下拉刷新开关。
- 步骤 02** 在页面的 js 文件中编写 onPullDownRefresh 函数，完成自己的下拉刷新逻辑。
- 步骤 03** 编写完下拉刷新逻辑代码后，主动调用 wx.stopPullDownRefresh 函数停止当前页面的下拉刷新。

我们首先来完成第一步。在 more-movie.json 文件中加入以下代码：

代码清单 10-31 配置下拉刷新开关

more-movie.json

```
{
  "enablePullDownRefresh": true
}
```

当在 more-movie.json 中加入以上代码后, more-movie 页面的下拉刷新就开启了。此时下拉 more-movie 页面将出现下拉效果。由于下拉刷新的等待标示默认是白色, 因此你无法明显地看到下拉刷新的等待状态标示。

我们可以修改等待标示的默认颜色, 在 more-movie.json 文件中添加一个配置项。

代码清单 10-32 配置下拉刷新图标的默认颜色

more-movie.json

```
{
  "enablePullDownRefresh": true,
  "backgroundTextStyle": "dark"
}
```

这样当再次下拉刷新 more-movie 页面时, 我们将看到等待标示, 如图 10-18 所示。

当然, 你也可以在 app.json 的 window 配置项中配置 backgroundTextStyle:"dark"选项, 这将导致所有页面的下拉刷新等待标示都变成 dark。

同理, 你也可以在 app.json 的 window 配置项中配置 enablePullDownRefresh:true 选项, 这将导致所有页面都开启下拉刷新。

接着我们来完成第二步。当页面打开下拉刷新开关后, 每当用户下拉页面都将触发执行页面的 onPullDownRefresh 函数。这就是小程序给我们编写下拉刷新逻辑的函数。在 more-movie.js 文件中编写 onPullDownRefresh 函数。



图 10-18 3 个点的下拉刷新等待标示

代码清单 10-33 onPullDownRefresh 函数

more-movie.js

```
onPullDownRefresh: function (event) {
  var refreshUrl = this.data.requestUrl +
    "?star=0&count=20";
  util.http(refreshUrl, this.processDoubanData);
}
```

整个函数的代码非常简单, 仅仅是再一次访问了豆瓣 API 并重新获取了最新的第 1 到第 20 条数据。

注意, 在函数中我们获取当前豆瓣 API 的 URL 地址时使用了 this.data.requestUrl (这个变

量是在 onLoad 函数中保存下来的)。所以,我们修改一下 more-movie 页面的 onLoad 函数,在 onLoad 函数中将当前访问的 URL 地址记录一下。

代码清单 10-34 保存当前访问的豆瓣 API 地址

more-movie.js

```
onLoad: function (options) {
  var category = options.category;
  var dataUrl = "";
  switch (category) {
    case "正在热映":
      dataUrl = app.globalData.doubanBase +
        "/v2/movie/in_theaters";
      break;
    case "即将上映":
      dataUrl = app.globalData.doubanBase +
        "/v2/movie/coming_soon";
      break;
    case "豆瓣 Top250":
      dataUrl = app.globalData.doubanBase + "/v2/movie/top250";
      break;
  }
  this.data.requestUrl = dataUrl;
  util.http(dataUrl, this.processDoubanData)
}
```

黑色加粗部分为新增代码。

编写完以上代码后,我们可以反复尝试下拉刷新 more-movie 页面。当然,从 UI 上是无法直接看到刷新效果的,因为豆瓣的电影数据不可能更新得非常频繁。我们可以尝试打开【Network】面板,观察一下每次下拉刷新后是否有向豆瓣发送请求,如果有请求发向豆瓣,就说明 onPullDownRefresh 函数成功触发了。

图 10-19 所示的示意图显示了 3 次下拉刷新 more-movie 页面后【Network】面板的请求发送情况。

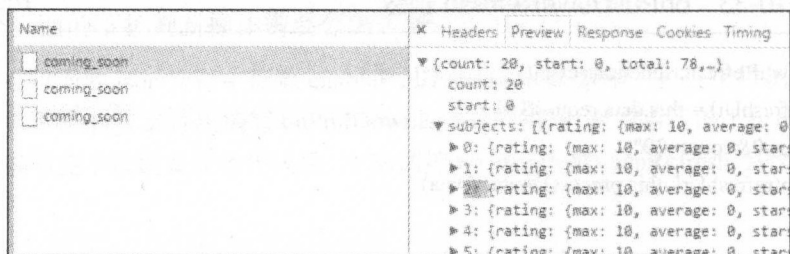


图 10-19 3 次刷新调用了 3 次豆瓣的 coming_soon 接口

最后,我们来完成下拉刷新“三部曲”中的第三步,主动停住页面刷新状态。停住页面刷新状态非常简单,在合适的时机调用 wx.stopPullDownRefresh 方法即可。

我们需要考虑的是在什么时候调用 `wx.stopPullDownRefresh()`。当然应该是在处理完豆瓣返回数据并再次调用 `this.setData` 重新绑定数据后调用 `wx.stopPullDownRefresh()` 函数。

在 `more-movie.js` 页面的 `processDoubanData` 方法中调用 `wx.stopPullDownRefresh()` 即可。

代码清单 10-35 结束下拉刷新状态

more-movie.js

```
processDoubanData: function (moviesDouban) {
  var movies = [];
  for (var idx in moviesDouban.subjects) {
    var subject = moviesDouban.subjects[idx];
    var title = subject.title;
    if (title.length >= 6) {
      title = title.substring(0, 6) + "...";
    }
    var temp = {
      stars: util.convertToStarsArray(subject.rating.stars),
      title: title,
      average: subject.rating.average,
      coverImageUrl: subject.images.large,
      movieId: subject.id
    }
    movies.push(temp)
  }
  this.setData({
    movies: movies
  });
  wx.stopPullDownRefresh();
}
```

黑色加粗部分为新增代码。

完成以上代码后，`more-movie` 页面的下拉刷新操作就全部完成了。下拉刷新三部曲是笔者总结的编写下拉刷新功能时的常见思维步骤，以供开发者参考。

10.18 在模拟器中可执行下拉刷新但在真机中无法执行下拉刷新的常见错误

在开启下拉刷新开关时，我们需要在页面的 `json` 文件中设置 `enablePullDownRefresh` 选项为 `true`。如果不小心将 `enablePullDownRefresh` 选项设置成了字符串“`true`”，而非 Boolean 类型的 `true`，就将导致在模拟器中可执行下拉刷新但在真机上（仅测试 iOS 系统）无法执行下拉刷新的现象。



虽然官方文档明确地指出 `enablePullDownRefresh` 配置选项的值类型是 `Boolean`，但真机行为同开发工具中行为不一致显然是不能接受的，这将增加测试成本，并有可能引起严重的 `bug`。

10.19 json 中的 `backgroundColor` 配置的是哪里颜色

官方文档中对 `backgroundColor` 这个配置项的解释是“配置窗口的背景色”，但这个文档中并没有明确解释窗口是什么、处于小程序的哪个部位。很多开发者都尝试设置 `backgroundColor`，但均无法看到设置效果，这会让人误以为这个配置选项是无效的，造成这个误解的主要原因是开发工具模拟器中的小程序和真机上小程序在执行下拉动作时有一些区别。

开发工具模拟器中的小程序是无法向下拉动的（如果不设置下拉刷新），但在真机上，无论你是否设置下拉刷新，导航栏以下的页面部分都可以向下拉动。拉动后，在导航栏和页面中间会有一块儿“空白”，`backgroundColor` 可以设置这块儿空白的颜色。

在模拟器中，可以通过设置下拉刷新看到这块儿区域。我们可以做一个测试来看看效果，在 `more-movie.json` 文件中增加一个 `backgroundColor` 配置选项。

代码清单 10-36 配置页面的 `backgroundColor`

`more-movie.json`

```
{
  "enablePullDownRefresh": true,
  "backgroundTextStyle": "dark",
  "backgroundColor": "black"
}
```

黑色加粗部分为新增代码。

增加以上配置选项后，再次下拉刷新 `more-movie.js` 页面，我们将看到如图 10-20 所示的刷新效果。

注意图中框起来的页面部分。这个部分的颜色将变成我们在 `more-movie.json` 文件中设置的 `backgroundColor` 颜色：黑色。

真机上的小程序存在一个问题，当配置导航栏颜色后，由于在真机上，即使不设置页面的下拉刷新，也会存在一个下拉的动作，露出一块儿白色的区域。这块儿白色的区域非常难看，我们可以通过设置页面的 `backgroundColor` 属性来配置这块儿区域的颜色。

首先在 `app.json` 中配置全局的窗口背景颜色。



图 10-20 `more-movie.js` 页面刷新效果

代码清单 10-38 配置全局窗口颜色

app.json

```
"window": {
  "navigationBarBackgroundColor": "#4A6141",
  "backgroundColor": "#4A6141"
}
```

接着，将 welcome 页面的窗口颜色单独配置。

代码清单 10-39 配置 welcome 页面的窗口颜色

welcome.json

```
{
  "navigationBarBackgroundColor": "#ECC0A8",
  "backgroundColor": "#ECC0A8"
}
```

通过以上设置将使小程序在真机上，即使下拉页面，也不会出现“空白”。

10.20 实现上滑加载更多数据

上滑加载更多数据又是另一个经典的 APP 操作。在本节中，我们将在小程序的 more-movie 页面中实现这个经典的操作。

目前的 more-movie 页面只能显示最多 20 条数据，因为豆瓣 API 最多只允许我们一次加载 20 条数据，若想显示更多的电影数据，则需要实现分步加载。在传统的 Web 网页上，我们通常是通过分页来实现显示更多数据。在移动端，更常见的操作是不考虑页码，通过不断地上滑页面来实现加载更多数据。

实现上滑加载更多数据的关键点在于何时触发“加载更多”这个操作。很明显，当页面“触底时”就可以执行“加载更多”这个操作了。

小程序在页面的 Page 中提供了一个 onReachBottom 函数。onReachBottom 将在每次页面上滑触底后触发执行。所以，我们只需要编写小程序提供的 onReachBottom 函数即可实现 more-movie 页面的上滑加载更多数据的功能。

在 more-movie.js 页面中新增 onReachBottom 函数。

代码清单 10-37 实现 onReachBottom 函数

more-movie.js

```
onReachBottom: function (event) {
  var totalCount = this.data.movies.length;
  //拼接下一组数据的 URL
  var nextUrl = this.data.requestUrl +
    "?start=" + totalCount + "&count=20";
```



```
util.http(nextUrl, this.processDoubanData)
```

```
}
```

以上代码是实现上滑加载更多操作的核心代码。在 `onReachBottom` 函数中，我们拼接了一个 `nextUrl` 作为取下一组电影数据的 URL。`start` 指定起始电影的条数，该数值等于当前页面已经绑定的电影条目条数，`count` 指定了最多取 20 条（当然不指定 `count` 也是可以的，因为默认最多取 20 条）。

现在的问题是，我们如何知道当前已经显示了多少条电影条目呢？

此时，我们需要调整下思路。我们将 `this.data.movies` 这个数组视作一个保存当前页面所有电影数据的数组。每当新增加 20 条数据后，我们都需要将新增的 20 条数据追加到这个 `this.data.movies` 数组中。

数据可以通过合并的形式追加到 `this.data.movies` 数组中，那么如何将新增的电影数据在 UI 上追加到已显示的电影后面呢？回顾一下我们在关于文章评论的 7.16 节中是如何将新的评论追加到已存在评论后面的。再次强调，要使用数据绑定的思维，而不要使用习惯的 DOM 思维来考虑问题。没有“追加”，只有重新绑定和渲染整个 `this.data.movies` 数组。所以，我们要做的就是将新加载的电影数据与已存在的电影数据合并在一起，并更新 `this.data.movies`。

修改 `more-movie.js` 文件中的 `processDoubanData` 函数。

代码清单 10-38 合并 movies 数组

`more-movie.js`

```
processDoubanData: function (moviesDouban) {
  var movies = [];
  for (var idx in moviesDouban.subjects) {
    var subject = moviesDouban.subjects[idx];
    var title = subject.title;
    if (title.length >= 6) {
      title = title.substring(0, 6) + "...";
    }
    var temp = {
      stars: util.convertToStarsArray(subject.rating.stars),
      title: title,
      average: subject.rating.average,
      coverageUrl: subject.images.large,
      movieId: subject.id
    }
    movies.push(temp)
  }
  var totalMovies = []
  totalMovies = this.data.movies.concat(movies);
  this.setData({
    movies: totalMovies
  });
};
```



```
wx.stopPullDownRefresh();
```

```
}
```

黑色加粗部分标记出了新增以及修改的代码。加粗部分代码将每次新增加的电影数据同已存在的电影数据合并在一起，并再次使用 `this.setData` 进行数据更新。`this.setData` 在做数据更新的同时也更新了 `this.data.movies` 这个变量，以确保 `this.data.movies` 变量永远记录的是当前页面的全部电影数据。

还有一个小问题是我们要考虑的，即当 `more-movie` 页面多次加载数据后如何处理下拉刷新这个操作。假设我们已经加载了 5 次共 100 条电影数据，那么此时去下拉刷新 `more-movie` 页面，理论上应该更新全部的 100 条数据，但是对于豆瓣电影没有必要这样处理。我们简化一下处理流程。无论 `more-movie` 页面有多少条电影数据，每当执行下拉刷新操作时都清空所有已存在的电影数据，重新加载最新的前 20 条数据。

根据以上思路，修改 `onPullDownRefresh` 函数。

代码清单 10-39 修改 `onPullDownRefresh` 函数

`more-movie.js`

```
onPullDownRefresh: function (event) {
  var refreshUrl = this.data.requestUrl +
    "?star=0&count=20";
```

```
//刷新页面后将页面所有初始化参数恢复到初始值
```

```
this.data.movies = [];
```

```
util.http(refreshUrl, this.processDoubanData);
```

```
}
```

黑色加粗部分为新增加的代码。当下拉刷新触发 `onPullDownRefresh` 函数后将 `this.data.movies` 清空。

完成以上代码后，现在每当我们在 `more-movie` 页面上滑触底后页面都将触发 `onReachBottom` 事件，从而实现加载更多电影数据。

10.21 动态设置导航栏 loading 图标

我们在前面实现了 `more-movie` 页面的下拉刷新电影数据与上滑加载更多电影数据的操作，但是整个加载数据的过程体验并不是那么好，总觉得缺少了点什么。

拿上滑加载更多电影数据来看，从触发加载数据到数据显示，整个过程没有等待提示，当数据加载完成后“突然”就显示了出来。

虽然我们建议目前的小程序不要使用太多的特效，但是必要的等待提示是用户最基本的心理预期，还是可以加上的。

在前面的章节中，我们学习过 `wx.showToast` 提示。`wx.showToast` 将在页面的中间位置显示一个模态或者非模态（`showToast` 的 `mask` 属性决定是模态还是非模态）的提示框。

笔者认为,对于 loading 状态提醒,用这种置于页面中心位置的侵入式提醒体验并不好。Loading 并不需要很强的信息通知,用户不需要阅读文字,只需要能够感觉到正在加载即可,所以使用“侵入式”的 `wx.showToast` 并不是太合适。我们希望有一个“非侵入式”的 loading 状态提醒,只让用户从心理上感觉到页面正在加载即可。

小程序提供的一个 API 方法 `wx.showNavigationBarLoading()` 非常适合用于 loading 状态提醒。通常来说, `wx.showNavigationBarLoading()` 和 `wx.hideNavigationBarLoading()` 是成对出现的,一个负责显示 loading 状态图标,一个负责隐藏 loading 状态图标。

图 10-21 显示了导航栏 loading 状态图标的效果。



图 10-21 导航栏 loading 状态图标

考虑一下,我们需要在 `more-movie` 页面的哪些操作中调用 `wx.showNavigationBarLoading()` 显示 loading 状态图标,又需要在什么地方调用 `wx.hideNavigationBarLoading()` 隐藏状态图标。

基本上,每次去豆瓣取数据时就应该显示 loading 状态,当数据处理完豆瓣电影数据并使用 `this.setData` 更新电影数据后应该调用 `wx.hideNavigationBarLoading()` 方法隐藏 loading 状态。

具体到 `more-movie` 页面,我们可以在 `onLoad` 初始化加载电影数据时、`onReachBottom` 滑动加载更多电影数据时以及 `onPullDownRefresh` 下拉刷新电影数据时调用 `wx.showNavigationBarLoading()` 显示 loading 状态图标,而在 `processDoubanData` 中处理完豆瓣数据后调用 `wx.hideNavigationBarLoading()` 隐藏状态图标。

在以上几个方法中分别添加 `wx.showNavigationBarLoading()` 与 `wx.hideNavigationBarLoading()` 方法。

代码清单 10-40 onLoad 函数中显示 loading

more-movie.js

```
onLoad: function (options) {
  //.....省略部分代码

  this.data.requestUrl = dataUrl;
  util.http(dataUrl, this.processDoubanData)
  //显示 loading 状态
  wx.showNavigationBarLoading();
}
```

代码清单 10-41 下拉刷新时显示 loading 状态

more-movie.js

```
onPullDownRefresh: function (event) {
  var refreshUrl = this.data.requestUrl +
    "?star=0&count=20";
  //刷新页面后将页面所有初始化参数恢复到初始值
  this.data.movies = [];
}
```




```

util.http(refreshUrl, this.processDoubanData);
//显示 loading 状态
wx.showNavigationBarLoading();
}

```

代码清单 10-42 加载更多时显示 loading 状态

more-movie.js

```

onReachBottom: function (event) {
    var totalCount = this.data.movies.length;
    //拼接下一组数据的 URL
    var nextUrl = this.data.requestUrl +
        "?start=" + totalCount + "&count=20";
    util.http(nextUrl, this.processDoubanData)
    //显示 loading 状态
    wx.showNavigationBarLoading();
}

```

代码清单 10-43 加载完成后隐藏 loading 状态

more-movie.js

```

processDoubanData: function (moviesDouban) {
    //.....省略部分代码
    this.setData({
        movies: totalMovies
    });
    wx.stopPullDownRefresh();
    //隐藏 loading 状态
    wx.hideNavigationBarLoading();
}

```

在以上几段代码中，黑色加粗部分为新增加的代码。

添加完以上代码后，开发者注意观察一下当正在加载数据时和数据加载完成后导航栏 loading 图标的显示和隐藏效果。

这里要特别说明以下 3 点：

第一，下拉刷新时可以不使用导航栏的 loading 状态图标，因为下拉刷新本身在页面的窗体部分就有一个 loading 状态图标，如图 10-18 所示。

第二，在以上代码中，我们没有考虑数据加载失败的情况。如果数据加载失败，程序就无法进入 processDoubanData 方法中，从而导致无法执行 wx.hideNavigationBarLoading() 方法，导航栏的 loading 图标一直显示。Orange Can 只是一个示例项目，如果完全模拟真实项目，需要写太多同小程序无关的代码，所以建议开发者在真实的项目中注意容错处理。“永远不要相信外部环境”是每个开发者要时刻牢记于心的真理。开发者可以尝试将隐藏 loading 图标的代码放在 wx.request 请求的 complete 函数中，这样无论数据加载是否成功都将隐藏 loading 图标。

第三，在 onLoad 函数中调用 wx.showNavigationBarLoading 函数设置页面导航栏是有风险



的, 原因我们已在 6.9.2 小节中解释过。当前版本没有问题, 并不代表未来版本不会有问题, 何况在 `onReady` 页面生命周期之后再操作界面元素是官方文档中明确说明的。所以建议开发者最好还是在页面的 `onReady` 函数中动态设置导航栏 `loading` 状态, 同官方文档的要求保持一致。下面给出在 `more-movie` 页面的 `onReady` 函数中设置 `loading` 状态的代码, 开发者可自行选择是否使用以下代码, 毕竟在 130400 版本中在 `onLoad` 函数中设置 `loading` 状态也是没有问题的。

代码清单 10-44 在 `onReady` 中设置 `loading` 状态

more-movie.js

```
onReady: function () {  
  wx.showNavigationBarLoading()  
}
```

从设置 `loading` 状态标示这个小示例中我们应该能体会到小程序到底简单在什么地方。你无须过多考虑 `loading` 应该怎么去提示, 小程序已经帮你“安排”好了。虽然这些固化的 API 让程序显得很没有个性, 但是小程序的特性就是用完即走。有时候牺牲一些个性化与美观度, 换来更高的开发效率与更低的发展成本是值得开发者考虑的。

当然, 如果你不满意小程序的 `loading`, 完全可以自行编码来实现自己理想的 `loading` 状态机制。

10.22 电影搜索

在本节中, 我们将实现电影搜索功能。电影搜索的效果图, 如图 10-22 所示。



图 10-22 电影搜索效果图

我们并没有选择使用一个新的页面来编写电影搜索的功能，电影搜索位于 movie 电影首页。当激活搜索时，电影首页的资讯面板信息被隐藏，电影搜索面板被显示；相反，当退出电影搜索时，电影搜索面板被隐藏，而电影资讯面板将被显示。

下面我们在电影首页中实现这个显示与隐藏的效果。在 movie.wxml 中添加并修改以下代码：

代码清单 10-45 加入电影搜索骨架代码

movie.wxml

```
<import src="movie-list/movie-list-tpl.wxml" />
<import src="movie-grid/movie-grid-tpl.wxml" />

<view class="search">
  <icon type="search" class="search-img" size="13" color="#405f80"></icon>
  <input type="text" placeholder="乘风破浪、西游伏妖篇"
    placeholder-class="placeholder" bindfocus="onBindFocus"
    bindconfirm="onBindConfirm" value="{{inputValue}}"/>
  <image wx:if="{{searchPanelShow}}" src="/images/icon/wx_app_xx.png" class="xx-img"
    catchtap="onCancellmgTap"></image>
</view>

<view class="container" wx:if="{{containerShow}}">
  <view class="movies-template">
    <template is="movieListTpl" data="{{...inTheaters}}"/>
  </view>
  <view class="movies-template">
    <template is="movieListTpl" data="{{...comingSoon}}"/>
  </view>
  <view class="movies-template">
    <template is="movieListTpl" data="{{...top250}}"/>
  </view>
</view>

<view class="search-panel" wx:if="{{searchPanelShow}}">
  <template is="movieGridTpl" data="{{...searchResult}}"/>
</view>
```

黑色加粗代码是添加并修改的代码。

首先我们添加了一个<view class="search">容器，在该容器的内部实现了一个 input 的搜索栏。placeholder 属性设置了 input 的默认占位文字；placeholder-class 属性指定了 placeholder 样式类的类名，该样式将在随后被添加到 movie.wxss 文件中。bindfocus 事件将实现鼠标或者手指激活 input 时显示搜索面板，bindconfirm 事件将实现提交搜索信息的功能。xx-img 将实现点击后关闭（隐藏）搜索面板。

同时，在<view class="container">这个电影资讯面板上增加一个属性 wx:if="{{containerShow}}", 这个属性也是用来控制电影资讯面板显隐状态的标识位。

最后，我们增加了一个<view class="search-panel">。这个面板就是电影搜索面板，主要用来显示搜索结果。

可以看到，在电影搜索面板中并没有编写任何代码，我们再一次地使用了 movie-grid-tpl 模板。所以，我们在代码的顶部再次引用了 movie-grid-tpl 模板。

添加完电影搜索的骨架代码后，我们来编写电影搜索的样式。在 movie.wxss 文件中添加以下同电影搜索相关的样式：

代码清单 10-46 添加电影搜索样式

movie.wxss

```
@import "movie-list/movie-list-tpl.wxss";
@import "movie-grid/movie-grid-tpl.wxss";

.container {
  background-color: #f2f2f2;
}

.movies-template {
  margin-bottom: 30rpx;
}

.search {
  background-color: #f2f2f2;
  height: 80rpx;
  width: 100%;
  display: flex;
  flex-direction: row;
}

.search-img {
  margin: auto 0 auto 20rpx;
}

.search input {
  height: 100%;
  width: 600rpx;
  margin-left: 20px;
  font-size: 28rpx;
}

.placeholder {
  font-size: 14px;
  color: #d1d1d1;
```




```

margin-left: 20rpx;
}

.search-panel{
  position: absolute;
  top: 80rpx;
}

.xx-img{
  height: 30rpx;
  width: 30rpx;
  margin: auto 0 auto 10rpx;
}

```

黑色加粗部分为新增代码。记得在代码的顶部引用 `movie-grid-tpl` 模板的样式。

当编写完以上代码后，保存并运行项目，发现电影首页的电影内容“消失”了。由于我们没有正确地在 `movie` 页面的 `js` 文件中设置搜索面板同电影资讯面板的显隐控制变量，因此电影资讯面板默认被隐藏了。

下面在 `movie.js` 的 `data` 变量中设置 `searchPanelShow` 与 `containerShow` 的默认值。

代码清单 10-47 设置默认的显示与隐藏状态

movie.js

```

data: {
  inTheaters: {},
  comingSoon: {},
  top250: {},
  containerShow: true,
  searchPanelShow: false,
  searchResult: {}
}

```

黑色加粗部分是新增代码。除了设置“`containerShow: true`”让电影资讯面板显示、“`searchPanelShow: false`”让搜索面板隐藏外，我们顺便将搜索结果的初始化变量 `searchResult` 也加入 `data` 变量中。

此时保存并运行代码，将出现如图 10-23 所示的效果。

以上代码实现了搜索面板与电影面板的初始化状态。现在，当用户激活 `input` 搜索栏准备输入关键字开始搜索时，我们需要显示搜索面板并隐藏电影资讯面板。已经注册在 `input` 组件上的 `onBindFocus` 事件将实现这个显隐切换效果。

在 `movie.js` 中添加 `onBindFocus` 事件响应函数。

代码清单 10-48 切换面板

movie.js

```
onBindFocus: function (event) {  
  this.setData({  
    containerShow: false,  
    searchPanelShow: true  
  })  
}
```

当用户激活 input 组件后将执行 onBindFocus 函数。这将隐藏电影资讯面板，并显示搜索面板，准备接受用户输入，如图 10-24 所示。



图 10-23 movie 电影首页加入搜索栏后的效果

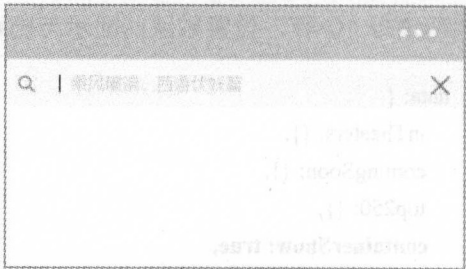


图 10-24 搜索面板被显示

用户通过点击图 10-24 中 input 搜索框右侧的 X 图片，可以关闭搜索面板并再次打开电影资讯面板。在 movie.js 中添加点击 X 图片的事件响应函数 onCancelImgTap。

代码清单 10-49 编写隐藏搜索面板代码

movie.js

```
onCancelImgTap: function (event) {  
  this.setData({  
    containerShow: true,  
    searchPanelShow: false,  
    searchResult: {},  
    inputValue:""  
  })  
}
```

在上述代码中，我们除了切换两个面板的显隐状态外，同时清空搜索结果 `searchResult`，保证下次再次进入搜索面板时，搜索面板不会记录上一次的搜索结果。将 `inputValue` 的值设置为空字符串将保证 `input` 组件所记录的用户输入值也一并被清空。

需要注意的是，`input` 组件的输入文本是无法设置字体的，因为在小程序中 `input` 组件是一个 `native` 组件，字体必须使用系统字体，所以无法设置 `font-family`。在真机上运行时，它也将被设置为真机系统的默认字体。

当用户输入关键字并按键盘上的“回车”或者点击真机上的“完成”后，小程序将触发 `input` 的 `bindconfirm` 事件，并执行已经注册在 `input` 上的事件响应函数 `onBindConfirm`。

在 `movie.js` 文件中添加 `onBindConfirm` 事件响应函数。

代码清单 10-50 响应搜索事件

movie.js

```
onBindConfirm: function (event) {
  var keyWord = event.detail.value;
  var searchUrl = app.globalData.doubanBase +
    "/v2/movie/search?q=" + keyWord;
  this.getMovieListData(searchUrl, "searchResult", "");
}
```

`onBindConfirm` 的代码非常简单，首先获取到用户的输入值 `keyWord`，这个值稍后将被附加到豆瓣电影搜索 API 的 URL 地址中。

豆瓣电影的搜索 API 地址为 `/v2/movie/search?q={keyword}`，我们只需要将获取到的 `keyWord` 附加在 API 地址中并调用 `this.getMovieListData` 方法即可。`this.getMovieListData` 方法早在编写 `movie` 页面的电影数据显示时已经编写好，这里只需要调用即可。

`this.getMovieListData` 接收 3 个参数，`searchUrl` 是豆瓣 API 的数据访问地址，`"searchResult"` 指定将处理完毕的豆瓣数据绑定到 `searchResult` 变量上，以提供给 `wxml` 显示数据：

```
<template is="movieGridTpl" data="{{...searchResult}}"/>
```

10.23 电影详情页面

下面我们来编写电影详情页面。

先思考一下电影详情页面的入口在哪里。任何一个显示电影封面（见图 10-25）的地方都应该能够点击进入到电影详情页面，这些入口遍布在电影首页、电影搜索、更多电影等多个地方。好在我们是使用 `template` 模板构建的电影页面结构体系，所有的电影单体元素都集中在 `movie-tpl` 这个 `template` 模板里。也就是说，只需要在 `movie-tpl` 模板上注册相应的点击事件即可。

在 10.4 节中，我们已经在代码清单 10-5 中注册了 `movie-tpl` 的点击事件 `onMovieTap`。现在要做的就是每个包含 `movie-tpl` 的页面中编写 `onMovieTap` 事件响应函数，并在事件响应函数中将页面跳转到 `movie-detail` 页面。

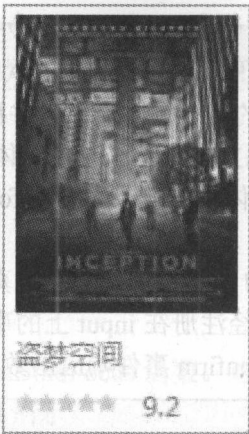


图 10-25 电影详情页面的入口元素

首先,新建 movie-detail 页面并注册这个页面。在 app.json 的 pages 数组下注册 movie-detail 页面。

代码清单 10-51 注册并新建 movie-detail 页面

app.json

```
"pages": [  
  "pages/welcome/welcome",  
  "pages/post/post",  
  "pages/post/post-detail/post-detail",  
  "pages/post/post-comment/post-comment",  
  "pages/movie/movie",  
  "pages/movie/more-movie/more-movie",  
  "pages/movie/movie-detail/movie-detail"  
]
```

黑色加粗部分为新增代码。保存后开发工具将自动在 pages/movie/movie-detail 目录下新建 movie-detail 页面的 4 个页面文件。

在以下两个使用过 movie-tpl 模板的页面 js 文件中编写 onMovieTap 事件响应函数：

- movie.js
- more-movie.js

代码清单 10-52 新增 onMovieTap 函数

movie.js

```
onMovieTap: function (event) {  
  var movieId = event.currentTarget.dataset.movieId;  
  wx.navigateTo({  
    url: "movie-detail/movie-detail?id=" + movieId  
  })  
}
```


代码清单 10-53 新增 onMovieTap 函数

more-movie.js

```
onMovieTap: function (event) {
  var movieId = event.currentTarget.dataset.movieId;
  wx.navigateTo({
    url: '../movie-detail/movie-detail?id=' + movieId
  })
}
```

注意以上两段代码所设置的 wx.navigateTo 的 url 参数值是不同的。我们同时还将所点击电影的 id 号附加在 url 中, 传递到 movie-detail 页面中。movie-detail 只有知道电影的 id 号才能够正确地加载电影详情数据。

从以上两段代码, 我们也可以再次看出小程序模板的缺陷: 模板不能运行自己的业务逻辑代码。onMovieTap 的业务逻辑本应当属于模板自身的业务逻辑, 但由于模板不可以运行 js 代码, 因此只能将本属于模板的业务逻辑编写在调用模板的页面 js 代码中。在 Orange Can 项目中, 我们幸运地只在两个页面引用了 movie-tpl 模板, 但如果有十几个地方引用了 movie-tpl 模板, 就不得不在这十几个地方一次编写上述代码。

当然, 可以将模板的业务代码编写成一个类似于 util 一样的模块, 并在其他页面里引用这个模块, 但 onMovieTap 里的代码终究还是属于 movie-tpl 的, 无论放在什么地方都不如放在模板自己的业务逻辑文件中。

编写完以上代码后, 无论是在 movie 页面、more-movie 页面还是在电影搜索的结果页面上, 点击任意电影都将跳转到 movie-detail 电影详情页面。

接下来, 我们需要编写 movie-detail 电影详情页面的骨架与样式。电影详情页面的效果如图 10-26 所示。



图 10-26 电影详情页面

10.24 电影详情页面的骨架和样式

本节我们开始着手编写 movie-detail 电影详情页面的骨架与样式代码。在 movie-detail.wxml 文件中添加以下代码:

代码清单 10-54 电影详情页面的骨架代码

movie-detail.wxml

```
<import src="../../stars/stars-tpl.wxml" />
<view class="container">
  <image class="head-img" src="{{movie.movieImg}}" mode="aspectFill" />
```

```

<view class="head-img-hover">
  <text class="main-title">{{movie.title}}</text>
  <text class="sub-title">{{movie.country + "·"+movie.year}}</text>
  <view class="like">
    <text class="highlight-font">
      {{movie.wishCount}}
    </text>
    <text class="plain-font">
      人喜欢
    </text>
    <text class="highlight-font">
      {{movie.commentCount}}
    </text>
    <text class="plain-font">
      条评论
    </text>
  </view>
</view>
<image class="movie-img" src="{{movie.movieImg}}" data-src="{{movie.movieImg}}"
catchtap="viewMoviePostImg"/>
<view class="summary">
  <view class="original-title">
    <text>{{movie.originalTitle}}</text>
  </view>
  <view class="flex-row">
    <text class="mark">评分</text>
    <template is="starsTpl" data="{{{stars:movie.stars, score:movie.score}}}" />
  </view>
  <view class="flex-row">
    <text class="mark">导演</text>
    <text>{{movie.director.name}}</text>
  </view>
  <view class="flex-row">
    <text class="mark">影人</text>
    <text>{{movie.casts}}</text>
  </view>
  <view class="flex-row">
    <text class="mark">类型</text>
    <text>{{movie.genres}}</text>
  </view>
</view>
<view class="hr"></view>
<view class="synopsis">

```



```

<text class="synopsis-font">剧情介绍</text>
<text class="summary-content">{{movie.summary}}</text>
</view>
<view class="hr"></view>
<view class="cast">
  <text class="cast-font">影人</text>
  <scroll-view class="cast-imgs" scroll-x="true" style="width:100%">
    <block wx:for="{{movie.castInfo}}" wx:for-item="item">
      <view class="cast-container">
        <image class="cast-img" src="{{item.img}}"></image>
        <text class="cast-name">{{item.name}}</text>
      </view>
    </block>
  </scroll-view>
</view>
</view>

```

注意，在 movie-detail 页面中我们同样使用了 stars-tpl 模板，记得在代码的顶部引入这个模板文件。

骨架代码中并没有什么新鲜的知识，唯一值得关注的是我们使用了一个 scroll-view 组件。这个组件用于横向展示多张演员图片，这些演员图片会“突破”手机尺寸的限制，在横向出现滚动条。图 10-27 展示了 scroll-view 的横向滚动效果图。

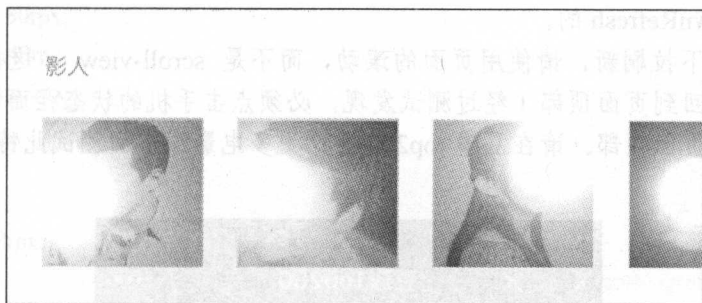


图 10-27 scroll-view 的横向滚动效果

scroll-view 组件的 scroll-x 和 scroll-y 分别设置组件的横向和纵向是否出现滚动条，如果想使用横向 scroll-view，就请设置 scroll-x="true"。

scroll-view 组件的横向排布要注意以下几个要点：

- 如果 scroll-view 下排列的多个子元素是块级元素（比如 view），就直接对 scroll-view 设置 display:flex 和 flex-direction:row，不会使子元素自动成为水平排列。如果不使用 scroll-view 而将容器元素换成 view，那么设置 display:flex 和 flex-direction:row 是可以使子元素自动成水平排列的。
- 如果想让 scroll-view 下的 view 元素水平排列，一种可行的方法是将子元素 view 设置为 inline-block 或者 inline-flex。

- 子元素有可能会换行的情况，需要在容器上设置 `white-space: nowrap`。

我们会在后面的章节中详细讲解 `scroll-view` 的一些高级用法，现在我们只需要掌握这些 `scroll-view` 的简单用法即可。

这里顺便提一下，`scroll-view` 中的 `bindscrolltolower` 事件也可以用来实现加载更多数据（类似于 `Page` 页面的 `onReachBottom`）。当 `scroll-view` 滚动到底部或者右边时将触发这个事件，实现原理请参考 `more-movie` 页面的 `onReachBottom` 函数。

实际上 `Orange Can` 项目最初在编写 `more-movie` 页面加载更多数据时也是使用的 `scroll-view` 的 `bindscrolltolower` 事件，但微信在 130400 版本更改了一些规则，官方文档中是这样描述的：在滚动 `scroll-view` 时会阻止页面回弹，所以在 `scroll-view` 中滚动是无法触发 `onPullDownRefresh` 的。

在 130400 版本之前，`scroll-view` 的滚动同 `onPullDownRefresh` 下拉刷新事件是不冲突的，但版本更新后若使用 `scroll-view` 则在 `scroll-view` 组件中下拉页面是无法触发 `onPullDownRefresh` 函数的。正是基于这个原因，笔者最后没有使用 `scroll-view` 来实现 `movie-grid-tpl` 模板，而是改为使用普通的 `view`，并使用 `Page` 的 `onReachBottom` 函数实现了加载更多数据的功能。

除此之外，`scroll-view` 还有以下几个需要注意的地方：

- 请勿在 `scroll-view` 中使用 `textarea`、`map`、`canvas`、`video` 组件。
- `scroll-into-view` 的优先级高于 `scroll-top`。
- 在滚动 `scroll-view` 时会阻止页面回弹，所以在 `scroll-view` 中滚动是无法触发 `onPullDownRefresh` 的。
- 若要使用下拉刷新，请使用页面的滚动，而不是 `scroll-view`，这样也能通过点击顶部状态栏回到页面顶部（经过测试发现，必须点击手机的状态栏而非小程序的导航栏才能返回页面顶部。请在豆瓣 `top250` 的“更多电影”页面测试此特性），如图 10-28 所示。

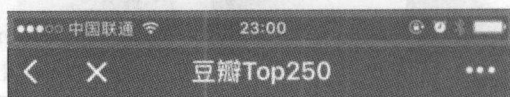


图 10-28 点击图中红色框部位才能返回页面顶部

关于 `scroll` 的更详细介绍和高级用法请参考 13.2 节，本节我们仅使用 `scroll-view` 的基本功能。

接着编写 `movie-detail` 页面的样式。将以下代码添加到 `movie-detail.wxss` 文件中：

代码清单 10-55 编写电影详情页面的样式

movie-detail.wxss

```
@import "../stars/stars-tpl.wxss";
```

```
.container{
  display: flex;
  flex-direction: column;
```




```
}

.head-img{
  width:100%;
  height: 320rpx;
  -webkit-filter:blur(20px);
}

.head-img-hover{
  width: 100%;
  height: 320rpx;
  position:absolute;
  top:0;
  left:0;
  display:flex;
  flex-direction: column;
}

.main-title{
  font-size: 19px;
  color:#fff;
  font-weight:bold;
  margin-top: 50rpx;
  margin-left: 40rpx;
  letter-spacing: 2px;
}

.sub-title{
  font-size: 28rpx;
  color:#fff;
  margin-left: 40rpx;
  margin-top: 30rpx;
}

.like{
  display:flex;
  flex-direction: row;
  margin-top: 30rpx;
  margin-left: 40rpx;
}

.highlight-font{
  color: #f21146;
```



```
font-size:22rpx;
margin-right: 10rpx;
}

.plain-font{
color: #666;
font-size:22rpx;
margin-right: 30rpx;
}

.movie-img{
height:238rpx;
width: 175rpx;
position: absolute;
top:160rpx;
right: 30rpx;
}

.summary{
margin-left:40rpx;
margin-top: 40rpx;
color: #777777;
}

.original-title{
color: #1f3463;
font-size: 24rpx;
font-weight: bold;
margin-bottom: 40rpx;
}

.flex-row{
display:flex;
flex-direction: row;
margin-bottom: 10rpx;
}

.mark{
margin-right: 30rpx;
white-space:nowrap;
color: #999999;
}
```



```
.hr{  
    margin-top:45rpx;  
    height:1px;  
    width: 100%;  
    background-color: #d9d9d9;  
}
```

```
.synopsis{  
    margin-left:40rpx;  
    display:flex;  
    flex-direction: column;  
    margin-top: 50rpx;  
}
```

```
.synopsis-font{  
    color:#999;  
}
```

```
.summary-content{  
    margin-top: 20rpx;  
    margin-right: 40rpx;  
    line-height:40rpx;  
    letter-spacing: 1px;  
}
```

```
.cast{  
    margin-left:40rpx;  
    display:flex;  
    flex-direction: column;  
    margin-top:50rpx;  
}
```

```
.cast-font{  
    color: #999;  
    margin-bottom: 40rpx;  
}
```

```
.cast-container{  
    display:inline-flex;  
    flex-direction: column;  
    margin-bottom: 50rpx;  
    margin-right: 40rpx;
```

```

width: 170rpx;
text-align:center;
white-space: normal;
}

.cast-imgs{
white-space: nowrap;
}

.cast-img{
width: 170rpx;
height: 210rpx;
}

.cast-name{
margin: 10rpx auto 0;
}

```

同样记得在代码的顶部引入 stars-tpl 模板的样式文件。

-webkit-filter:blur(20px)实现了图片模糊的效果。

由于现在并没有编写页面的逻辑文件，因此我们暂时无法看到页面的显示效果。

10.25 编写电影详情页面的业务逻辑代码

现在编写 movie-detail 电影详情页面的业务逻辑代码。

在 movie-detail.js 文件中添加以下代码：

代码清单 10-56 编写电影详情页面业务逻辑

movie-detail.js

```

var util = require('../../util/util.js')

var app = getApp();
Page({
  data: {
    movie: {}
  },
  onLoad: function (options) {
    var movieId = options.id;
    var url = app.globalData.doubanBase +
      "/v2/movie/subject/" + movieId;

    util.http(url, this.processDoubanData)
  },

  processDoubanData:function(data) {

```




```

    if (!data) {
        return;
    }

    var director = {
        avatar: "",
        name: "",
        id: ""
    }

    if (data.directors[0] != null) {
        if (data.directors[0].avatars != null) {
            director.avatar = data.directors[0].avatars.large
        }
        director.name = data.directors[0].name;
        director.id = data.directors[0].id;
    }

    var movie = {
        movieImg: data.images ? data.images.large : "",
        country: data.countries[0],
        title: data.title,
        originalTitle: data.original_title,
        wishCount: data.wish_count,
        commentCount: data.comments_count,
        year: data.year,
        genres: data.genres.join("、"),
        stars: util.convertToStarsArray(data.rating.stars),
        score: data.rating.average,
        director: director,
        casts: util.convertToCastString(data.casts),
        castsInfo: util.convertToCastInfos(data.casts),
        summary: data.summary
    }

    this.setData({
        movie: movie
    })
}
})

```

整个代码逻辑非常简单，仅仅是获取数据并处理数据。豆瓣 API 获取电影详情数据的 URL 为 `/v2/movie/subject/{subjectID}`。

需要注意的是，豆瓣的许多电影由于年代久远而导致数据有所缺失，这会造成很多“空值”情况。有些空值只会让数据无法显示，但程序不会报错；有些空值则会直接导致程序终端运行。对于第二种空值，我们一定要做容错处理。

导致程序终端的空值多是由于对象是 `null`，而我们却尝试去读取空对象的属性，这样的行为导致程序报错，无法继续运行。如果是字符串 `null`，那么绑定一个空字符串程序是不会报错的。

豆瓣电影的数据结构非常复杂，无法一次将所有的属性都做容错处理，所以建议遇到错误时再回头编写容错代码。比如上述代码中对 `directors` 的属性处理就属于容错性处理。

在以上代码中，又调用了两个 `util` 函数：`util.convertToStarsArray` 和 `util.convertToCastInfos`。现在在 `util` 模块中编写这两个函数。

代码清单 10-57 编写 `convertToStarsArray`

util.js

```
//将数组转换为以 "/" 分隔的字符串
function convertToCastString(casts) {
  var castsjoin = "";
  for (var idx in casts) {
    castsjoin = castsjoin + casts[idx].name + " / ";
  }
  return castsjoin.substring(0, castsjoin.length - 2);
}
```

代码清单 10-58 编写 `convertToCastInfos`

util.js

```
function convertToCastInfos(casts) {
  var castsArray = []
  for (var idx in casts) {
    var cast = {
      img: casts[idx].avatars ? casts[idx].avatars.large : "",
      name: casts[idx].name
    }
    castsArray.push(cast);
  }
  return castsArray;
}
```

最后记得使用 `module.exports` 将这两个函数输出。

代码清单 10-59 输出方法

util.js

```
module.exports = {
  getDiffTime: getDiffTime,
  convertToStarsArray: convertToStarsArray,
  http: http,
  convertToCastString: convertToCastString,
  convertToCastInfos: convertToCastInfos
}
```



黑色加粗的代码是新增代码。

保存并运行代码，点击任意一张电影海报都可以将页面跳转到 movie-detail 页面，并正确显示电影详情数据。

10.26 预览电影海报

豆瓣的电影海报是比较精美的，我们可以尝试在 movie-detail 电影详情页面新增一个预览电影海报的功能。

点击 movie-detail 中悬浮在头图上的电影海报将打开一张电影海报大图，如图 10-29 所示。红色框所框选的区域就是触发点击事件的区域。



图 10-29 点击图中红色框部分将打开图片预览

在之前编写文章评论时我们已经讲解过图片预览。同时，在编写 movie-detail.wxml 文件时已经在这个 image 上注册了 viewMoviePostImg 事件。现在我们只需要实现 viewMoviePostImg 事件响应函数即可。

在 movie-detail.js 文件中添加 viewMoviePostImg 函数。

代码清单 10-60 编写图片预览函数

movie-detail.js

```
//预览电影海报
viewMoviePostImg: function (event) {
  var src = event.currentTarget.dataset.src;
  wx.previewImage({
    current: src,
    urls: [src]
  })
}
```

保存代码，尝试点击小海报，将打开一张大图。

电影详情页面是非常值得分享的内容，开发者可自行用我们之前讲到的分享 API 实现电影详情页面的分享功能。

10.27 设置电影页面的导航栏标题

所有配置或者是动态设置导航栏标题的方法在之前的章节中已详细讲解过，本节就不赘述了。下面直接给出配置代码。

首先，配置 movie 电影首页的导航栏标题，在 movie.json 文件中新增以下代码：

代码清单 10-61 配置电影首页导航栏标题

movie.json

```
{
  "navigationBarTextStyle": "white",
  "navigationBarTitleText": "光影"
}
```

接着，设置 more-movie 更多电影页面的导航栏标题。在 more-movie.js 页面中添加以下代码，以保证 more-movie 页面的导航栏标题被设置为当前电影的电影分类：

代码清单 10-62 保存当前电影类型

more-movie.js

```
onLoad: function (options) {
  var category = options.category;
  this.data.navigateTitle = category;
}
```

//.....省略若干行代码

以上代码黑色加粗部分为新增代码。将 category 电影类型暂时保存在 data 变量中，再在 more-movie.js 的 onReady 函数中读取这个变量并动态设置导航栏标题。

代码清单 10-63 动态设置导航栏标题

more-movie.js

```
onReady: function () {
  wx.setNavigationBarTitle({
    title: this.data.navigateTitle
});
  wx.showNavigationBarLoading()
}
```

最后，我们还需要将电影详情页面的导航栏标题动态设置为当前显示电影的电影标题。在 movie-detail.js 文件的 processDoubanData 中新增一小段代码。



代码清单 10-64 动态设置导航栏标题

movie-detail.js

```
processDoubanData: function(data) {
```

```
//.....省略若干行代码
```

```
    this.setData({
      movie: movie
    });
```

```
    wx.setNavigationBarTitle({
      title: data.title
    });
```

```
}
```

黑色加粗部分为新增代码。

编写完以上代码后，3 个电影页面的导航栏标题就全部设置完成了。

第 11 章

设 置

本章我们将学习小程序中众多的 API，这些 API 包括获取用户基本信息、获取系统信息、获取网络状态、获取当前位置信息与速度信息、使用微信内置地图查看当前位置、使用监听罗盘数据接口制作一个建议的指南针、经典小功能摇一摇、二维码扫描以及下载文档并预览文档等。

```
ready function() {  
  wx.navigateTo({  
    url: this.data.navigateTo  
  })  
}
```

11.1 设置页面

一个项目是不可能将小程序全部功能都使用到的，所以我们特意在 Orange Can 项目增加了一个“设置”页面。这个页面将集成很多零散的小程序功能，比如获取用户微信信息、系统信息、网络状态、指南针、获取当前位置、摇一摇、扫描二维码等功能。不同于文章和电影页面，“设置”页面的功能类似于小程序 API 的示例集合，虽然没有实际意义，但是对于了解和学习这些 API 的使用方法有极大帮助。

首先，在 app.json 的 pages 数组下新建 setting（设置）页面的路径。

代码清单 11-1 新建 setting 页面

app.json

```
"pages": [  
  "pages/welcome/welcome",  
  "pages/post/post",  
  "pages/post/post-detail/post-detail",  
  "pages/post/post-comment/post-comment",  
  "pages/movie/movie",  
  "pages/movie/more-movie/more-movie",  
  "pages/movie/movie-detail/movie-detail",  
  "pages/setting/setting"  
]
```

黑色加粗部分是新增的 setting 页面路径。

setting 同 post 和 movie 一样，也属于 tab 选项栏中的一项，将 setting 页面加入到 tab 选项卡中。

代码清单 11-2 在 tab 选项卡中加入 setting 页面

app.json

```
"list": [  
  {  
    "pagePath": "pages/post/post",  
    "text": "文字",  
    "iconPath": "images/icon/wx_app_news.png",  
    "selectedIconPath": "images/icon/wx_app_news@HL.png"  
  },  
  {  
    "pagePath": "pages/movie/movie",  
    "text": "光影",  
    "iconPath": "images/icon/wx_app_movie.png",  
    "selectedIconPath": "images/icon/wx_app_movie@HL.png"  
  }  
]
```



```

{
  "pagePath": "pages/setting/setting",
  "text": "设置",
  "iconPath": "images/icon/wx_app_setting.png",
  "selectedIconPath": "images/icon/wx_app_setting@HL.png"
}

```

这样，Orange Can 项目的 tab 选项卡将有 3 个选项：post、movie 和 setting。来看看 setting 页面的效果图，如图 11-1 所示。

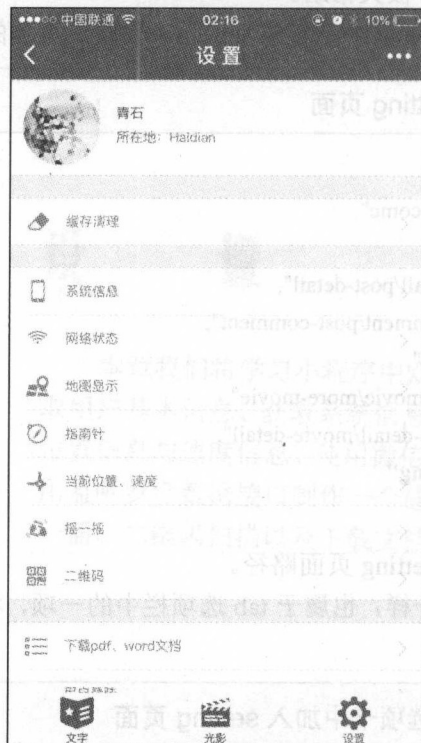


图 11-1 设置页面效果图

顶部显示当前用户的微信头像、微信昵称等信息；下面紧跟着的分别是缓存面板、设备面板、API 面板以及其他杂项面板。每个面板下包含若干种小程序常用功能，点击可执行或演示相应的功能。

下面编写 setting 页面的骨架结构。

代码清单 11-3 setting 页面的骨架

setting.wxml

```

<view class="container">
  <view class="category-item personal-info">
    <view class="user-avatar">
      <image src="{{userInfo.avatarUrl}}"></image>
    </view>
  </view>
</view>

```



```

</view>
<view class="user-name">
  <view class="user-nickname">
    <text>{{userInfo.nickName}}</text>
  </view>
  <view class="user-location">
    <text>所在地: {{userInfo.city}}</text>
  </view>
</view>
</view>

```

<!--缓存面板-->

```

<view class="category-item">
  <block wx:for="{{cache}}">
    <view class="detail-item" catchtap="{{item.tap}}">
      <image src="{{item.iconurl}}"></image>
      <text>{{item.title}}</text>
      <view class="detail-item-btn"></view>
    </view>
  </block>
</view>

```

<!--设备面板-->

```

<view class="category-item">
  <block wx:for="{{device}}">
    <view class="detail-item" catchtap="{{item.tap}}">
      <image src="{{item.iconurl}}"></image>
      <text>{{item.title}}</text>
      <view class="detail-item-btn"></view>
    </view>
  </block>
</view>

```

<!-- API 面板-->

```

<view class="category-item">
  <block wx:for="{{api}}">
    <view class="detail-item" catchtap="{{item.tap}}">
      <image src="{{item.iconurl}}"></image>
      <text>{{item.title}}</text>
      <view class="detail-item-btn"></view>
    </view>
  </block>
</view>

```



```

//其他杂项面板
<view class="category-item">
  <block wx:for="{{others}}">
    <view class="detail-item" catchtap="{{item.tap}}">
      <image src="{{item.iconurl}}"></image>
      <text>{{item.title}}</text>
      <view class="detail-item-btn"></view>
    </view>
  </block>
</view>

</view>

```

我们没有直接将每个面板的子项内容编码在 wxml 文件中。由于每个面板下的子项较多，且有可能经常添加子项，因此没有直接将子项“硬编码”在 wxml 文件中，而是选用了一种“配置式”的编写方法。每个项目下的子项内容都将在 js 文件中“配置”，然后通过数据绑定和列表渲染动态的填充到 wxml 中。

目前，由于我们还没有编写 setting 的 js 文件，列表渲染的各个数组都是“空值”状态，因此 setting 页面不会显示任何内容。

接着，编写 setting 页面的样式，在 setting.wxss 文件中加入以下代码：

代码清单 11-4 setting 页面的样式

setting.wxss

```

.container {
  background-color: #efff4;
  width: 100%;
  height: 100%;
  flex-direction: column;
  display: flex;
  align-items: center;
  min-height: 100vh;
}
.category-item {
  width: 100%;
  margin: 20rpx 0;
  border-top: 1rpx solid #d9d9d9;
  border-bottom: 1rpx solid #d9d9d9;
  background-color: #fff;
}
.category-item.personal-info {
  height: 130rpx;
  display: flex;

```



```

padding: 20rpx 0;
}
.category-item.personal-info .user-avatar {
margin: 0 30rpx;
width: 130rpx;
height: 130rpx;
position: relative;
}
.category-item.personal-info .user-avatar image {
vertical-align: top;
width: 100%;
height: 100%;
position: absolute;
top: 0;
left: 0;
border-radius: 2rpx;
}
.category-item.personal-info .user-name {
margin-right: 30rpx;
flex: 1;
padding-top: 10rpx;
}
}
.detail-item {
display: flex;
margin-left: 30rpx;
border-bottom: 1px solid RGBA(217, 217, 217, .4);
height: 85rpx;
align-items: center;
}
}
.detail-item:last-child {
border-bottom: none;
}
}
.detail-item image {
height: 40rpx;
width: 40rpx;
}
}
.detail-item text {
color: #7F8389;
font-size: 24rpx;
flex: 1;
margin-left: 30rpx;
}
}

```



```

.detail-item .detail-item-btn{
  width: 50rpx;
  color: #d9d9d9;
  height: 40rpx;
  margin-right: 20rpx;
  text-align: center;
}
.detail-item .detail-item-btn::after{
  display: inline-block;
  content:"";
  width: 16rpx;
  height: 16rpx;
  color: #d9d9d9;
  margin-top: 8rpx;
  border:3rpx #d9d9d9 solid;
  border-top-color:transparent;
  border-left-color:transparent;
  transform: rotate(-45deg);
}

```

最后，编写 setting.js 文件，在 js 里加入我们的配置。

代码清单 11-5 setting 页面的配置

setting.js

```

Page({
  data: {
    cache: [
      { iconurl: '/images/icon/wx_app_clear.png', title: '缓存清理', tap: 'clearCache' }
    ],
    device: [
      { iconurl: '/images/icon/wx_app_cellphone.png', title: '系统信息', tap: 'showSystemInfo' },
      { iconurl: '/images/icon/wx_app_network.png', title: '网络状态', tap: 'showNetWork' },
      { iconurl: '/images/icon/wx_app_location.png', title: '地图显示', tap: 'showMap' },
      { iconurl: '/images/icon/wx_app_compass.png', title: '指南针', tap: 'showCompass' },
      { iconurl: '/images/icon/wx_app_lonlat.png', title: '当前位置、速度', tap: 'showLonLat' },
      { iconurl: '/images/icon/wx_app_shake.png', title: '摇一摇', tap: 'shake' },
      { iconurl: '/images/icon/wx_app_scan_code.png', title: '二维码', tap: 'scanQRCode' }
    ],
    api: [
      { iconurl: '/images/icon/wx_app_list.png', title: '下载 pdf、word 文档', tap: 'downloadDocumentList' },
      { iconurl: "", title: '用户登录', tap: 'login' },
      { iconurl: "", title: '校验用户信息', tap: 'check' },
      { iconurl: "", title: '获取用户加密信息', tap: 'decrypted' },
      { iconurl: "", title: '模板消息', tap: 'tplMessage' },

```




```

{ iconurl: "", title: '微信支付', tap: 'wxPay' }
],
others: [
  { iconurl: "", title: 'wx:key 示例', tap: 'showWxKeyDemo' },
  { iconurl: "", title: 'scroll-view 高级用法演示', tap: 'showScrollViewDemo' }
]
})

```

以上代码在 setting.js 的 data 变量下加入了 4 个数组，分别对应 wxml 中的缓存面板数组、设备面板数组、API 面板数组以及其他杂项面板数组。每个数组子项都包含 iconurl、title 和 tap 3 个属性。iconurl 代表子项图标图片的路径，title 代表子项的标题名，tap 是点击子项后所执行函数的函数名。

这样，如果以后我们想新增一个子项，只需要在 js 文件中增加配置即可。编写完以上代码后，保存并运行项目，setting 页面将呈现如图 11-2 所示的效果。



图 11-2 setting 页面

顺便在 setting.json 文件中加入导航栏标题的配置。

代码清单 11-6 setting 页面导航栏配置

setting.json

```

{
  "navigationBarTextStyle": "white",
  "navigationBarTitleText": "设置"
}

```

我们将在后面的章节中逐步完成每一个子项的功能。

11.2 获取用户基本信息

目前 setting 页面的顶部还是空白的，我们需要在小程序中获取用户的基本资料，并显示在 setting 页面的顶部。小程序提供了一个 `wx.getUserInfo(OBJECT)` 方法来获取用户的信息。

用户信息分为用户基本信息和用户 `openId`、`UnionId`。基本信息是明文的，而 `openId` 和 `UnionId` 是加密数据。这两种类型的数据都由 `wx.getUserInfo(OBJECT)` 方法返回。

用户基本信息包括以下 7 项：

- `avatarUrl` 用户微信头像的 URL 地址。
- `city` 城市。
- `country` 国家。
- `gender` 性别，1 表示男，2 表示女，0 表示未知。
- `language` 语言区域。
- `nickName` 昵称。
- `province` 省份。

在小程序中，用户的基本信息可以轻易获得，他们是明文的、不加密的。但 `openId` 和 `UnionId` 是加密的。什么是 `openId` 和 `UnionId` 呢？可以将 `openId` 和 `UnionId` 理解为用户在微信应用中的 id 号。他们的区别是：`openId` 只代表用户在某个微信应用下的 id 号；而 `UnionId` 是跨应用的，同一用户在同一开发者的多个应用里，`UnionId` 是唯一的。

官方文档对于 `UnionId` 的描述非常清楚：如果开发者拥有多个移动应用、网站应用和公众账号（包括小程序），可通过 `UnionId` 区分用户的唯一性，因为只要是同一个微信开放平台账号下的移动应用、网站应用和公众账号（包括小程序），用户的 `UnionId` 就是唯一的。换句话说，同一用户在同个微信开放平台下的不同应用中，`UnionId` 是相同的。

所以，`openId` 不能跨应用，如果要在多应用间统一用户身份，请使用 `UnionId`。这里要注意，在小程序中使用 `UnionId` 首先需要前往微信开放平台绑定小程序。详情请参考官方文档：<https://mp.weixin.qq.com/debug/wxadoc/dev/api/open.html#wxgetuserinfoobject>。

在小程序中获取用户的基本信息是非常简单的，但想拿到 `openId` 和 `UnionId` 却不是那么容易。本节先来学习如何获取用户的基本信息，获取加密信息在后面的章节中讲解。

下面来看 `wx.getUserInfo(OBJECT)` 方法的使用。`wx.getUserInfo` 的 `OBJECT` 参数有 3 个回调函数：`success`、`fail` 和 `complete`。在这 3 个回调函数中，重点介绍 `success`。`success` 方法的返回值有以下 5 个：

- `userInfo` 用户基本信息对象，不包含 `openId` 等敏感信息。
- `rawData` 不包含敏感信息的基本信息字符串，通常用来计算签名，防止从微信返回的用户信息被篡改。
- `signature` 使用 `sha1(rawData + sessionkey)` 得到字符串，用于校验用户信息。
- `encryptedData` 包括敏感数据在内的完整用户信息的加密数据。



- iv 加密算法的初始向量。

如果不是非常了解微信的加解密及身份认证机制，建议开发者先不要过于纠结 signature、encryptedData、iv 这 3 个返回值，目前还不需要使用这 3 个参数。在后面的章节中会具体讲解这 3 个返回值的概念以及使用方法，本节只需要使用 userInfo 返回值。

现在，我们在 Orange Can 项目中获取用户的明文信息。首先，在 app.js 文件中的.globalData 对象下新增一个全局变量 g_userInfo，用来记录用户的基本信息。当我们需要在其他页面使用用户信息时，读取这个全局变量即可。

代码清单 11-7 新增 g_userInfo 全局变量

app.js

```
globalData: {
  g_isPlayingMusic: false,
  g_currentMusicPostId: null,
  doubanBase: "https://api.douban.com",
  g_userInfo: null
}
```

黑色加粗部分代码为新增代码。

随后，在 app.js 文件中添加一个 _getUserInfo 方法，用来获取用户的基本信息。注意这个方法位于 app.js 文件的 App 方法内部。

代码清单 11-8 获取用户基本信息

app.js

```
_getUserInfo: function () {
  var userInfoStorage = wx.getStorageSync('user');
  if (!userInfoStorage) {
    // 如果缓存中没有用户信息，那么获取用户信息
    var that = this;
    wx.login({
      success: function () {
        wx.getUserInfo({
          success: function (res) {
            that.globalData.userInfo = res.userInfo
            //将用户的基本信息保存到缓存中
            wx.setStorageSync('user', res.userInfo)
          },
          fail: function (res) {
            console.log(res);
          }
        })
      }
    })
  }
}
```



```
else {  
    // 如果缓存中已经存在用户的基本信息，那么将信息保存到全局变量中  
    this.globalData.userInfo = userInfoStorage;  
}  
}
```

解释一下上述代码。为了避免每次启动小程序时都要在微信服务器中加载用户的基本信息，我们将用户信息保存到缓存中。这样，每次启动小程序时先去查看缓存有没有数据，如果没有，就调用 `wx.getUserInfo` 获取用户信息并保存到缓存和全局变量 `g_userInfo` 中；如果有，就直接将用户信息保存到全局变量 `g_userInfo` 里。

细心的开发者可能注意到，我们并不是直接调用 `wx.getUserInfo(OBJECT)` 方法，而是先调用 `wx.login`，在 `wx.login` 调用成功后再继续调用 `wx.getUserInfo`。之所以在 `wx.login` 调用成功后才调用 `wx.getUserInfo`，是因为官方文档明确指出了获取用户信息需要先调用 `wx.login`。同时，在官方的示例项目中也是先调用 `wx.login`。至于 `wx.login` 有什么作用，我们会在后面的章节中讲到，开发者可“依葫芦画瓢”，照做即可。

事实上，如果你只想获取用户的明文基本信息，那么不调用 `wx.login`，直接调用 `wx.getUserInfo` 也可以获取用户信息。所以，是否在调用 `wx.getUserInfo` 前调用 `wx.login` 请开发者自行决定。

当第一次运行以上代码并调用 `wx.getUserInfo` 时，小程序会弹出如图 11-3 所示的提示窗口，让用户选择是否授权获取用户的基本信息。

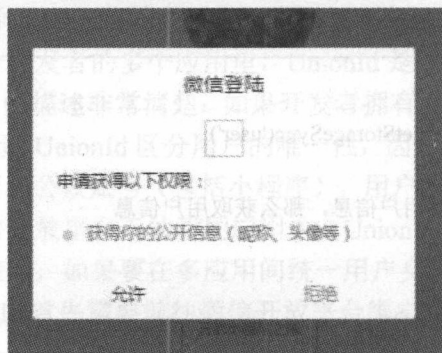


图 11-3 用户授权获取信息

用户点击“允许”后，才会执行 `wx.getUserInfo` 的 `success` 回调函数，如果点击“拒绝”，将执行 `fail` 回调函数。

有些开发者在测试时可能永远不会弹出这个授权窗口。因为如果当前项目没有 `appId`，那么用户信息是由微信模拟的信息，并没有进行真实的获取用户信息流程，所以不会出现这个窗口，且开发工具会有一个警告提示，如图 11-4 所示。

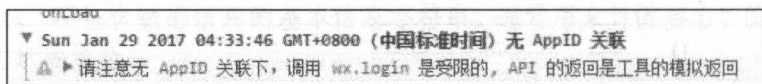


图 11-4 没有 `appId` 时是开发工具模拟的用户数据

当然，这个用户数据也是真实的用户数据，因为你在使用开发工具时必须扫描二维码登录，这样开发工具就能知道你的微信身份，自然可以模拟返回你的用户数据。但模拟数据相比于真实的用户数据缺少 3 个属性，即 `signature`、`encryptedData` 和 `iv`。也就是说，模拟数据只有明文信息而没有加密信息。

将用户信息保存到缓存中有一个缺点，就是没办法实时更新用户信息（比如用户更改了自己的微信资料），但这是具体业务的问题，需要开发者在实际项目编写中灵活处理。

现在，可以在 `setting` 页面中通过访问 `g_userInfo` 全局变量获取并显示用户的基本信息，需要在 `setting` 页面的 `onLoad` 方法中添加以下代码。

代码清单 11-9 获取用户信息并绑定

setting.js

```
var app=getApp();

onLoad: function () {
  this.setData({
    userInfo: app.globalData.g_userInfo
  })
}
```

注意，在使用全局变量前，需要先调用 `getApp()` 方法获取小程序的 `App` 对象。保存并运行代码，`setting` 页面将正确显示用户的头像、昵称等信息。

11.3 数据缓存的异步操作

为了可以在真机上清除数据缓存，我们在设置里增加一个清除数据缓存的选项。当用户点击这个选项时，首先会弹出一个 `modal` 模态窗口，如果用户点击【确定】，就可以清除用户的数据缓存。

在 5.8.2 小节中，我们提到过所有缓存类 API 除了有一组同步方法外，还有一组异步方法。在本小节中，尝试使用异步的 `wx.clearStorage()` 方法清除用户的数据缓存。

`setting` 页面有许多选项需要使用 `modal` 窗口让用户确认操作，所以我们在 `setting.js` 文件中添加一个显示 `modal` 的公共方法。

代码清单 11-10 showModal 公共方法

setting.js

```
//显示模态窗口
showModal: function (title, content, callback) {
  wx.showModal({
    title: title,
    content: content,
    confirmColor: '#1F4BA5',
    cancelColor: '#7F8389',
```



```

    success: function (res) {
      if (res.confirm) {
        callback && callback();
      }
    }
  })
}

```

showModal 方法只是对 MINA 框架提供的 wx.showModal 做了一次封装, 设置了通用的颜色。showModal 方法还接收了一个回调函数 callback, 用户点击【确定】后执行这个回调函数。

接着编写清除数据缓存的事件响应函数。从 data 中的 cache 配置来看, 清除数据缓存的事件响应函数为 clearCache。下面在 setting.js 文件中新增一个 clearCache 方法。

代码清单 11-11 清楚用户的数据缓存

setting.js

```

// 缓存清理
clearCache: function () {
  this.showModal('缓存清理', '确定要清除本地缓存吗?', function () {
    wx.clearStorage({
      success: function (msg) {
        wx.showToast({
          title: "缓存清理成功",
          duration: 1000,
          mask: true,
          icon: "success"
        })
      },
      fail: function (e) {
        console.log(e)
      }
    })
  });
}

```

显然, 异步缓存操作方法在写法上比同步要麻烦一些。要注意的是, 以上只是一个简单的异步函数操作示例, 所以看起来并不是那么“烦琐”, 但在真实的项目中, 往往有着非常复杂的业务逻辑。在 js 中, 异步非常容易形成所谓的“回调地狱”, 也就是在回调函数中连续嵌套调用回调函数, 这会让代码的可阅读性变得非常糟糕。异步函数又一直存在着难以调试的问题。

关于同步和异步的选择, 需要根据实际情况来决定, 笔者的建议是优选选择同步方法。当同步方法的体验和效率太差或者同步方法无法解决问题时再考虑使用异步方法。

至此, 完成了 setting 页面的第一个小功能。



11.4 获取系统信息

系统信息的显示需要新建一个 device 子页面。下面在 app.json 文件的 pages 数组下新增 device 页面的路径。

代码清单 11-12 新增 device 页面

app.json

```
"pages": [
  "pages/welcome/welcome",
  "pages/post/post",
  "pages/post/post-detail/post-detail",
  "pages/post/post-comment/post-comment",
  "pages/movie/movie",
  "pages/movie/more-movie/more-movie",
  "pages/movie/movie-detail/movie-detail",
  "pages/setting/setting",
  "pages/setting/device/device"
]
```

当我们点击【系统信息】时，setting 页面要跳转到 device 子页面，需要在 setting.js 中新增一个【系统信息】的点击事件响应函数。

代码清单 11-13 showDeviceInfo 函数

setting.js

```
//显示系统信息
showSystemInfo:function() {
  wx.navigateTo({
    url:'device/device'
  });
}
```

点击【系统信息】后，页面将跳转到 device 子页面。下面来编写 device 子页面的骨架和样式。在 device.wxml 和 device.wxss 中分别加入骨架代码和样式代码。

代码清单 11-14 device 子页面的骨架

device.wxml

```
<view class="container">
  <view class="category-item">
    <block wx:for="{{phoneInfo}}">
      <view class="detail-item">
        <text>{{item.key}}</text>
        <text>{{item.val}}</text>
      </view>
    </block>
  </view>
</view>
```



```

    </view>
  </block>
</view>
<view class="category-item">
  <block wx:for="{{softInfo}}">
    <view class="detail-item">
      <text>{{item.key}}</text>
      <text>{{item.val}}</text>
    </view>
  </block>
</view>
<view class="category-item">
  <block wx:for="{{screenInfo}}">
    <view class="detail-item">
      <text>{{item.key}}</text>
      <text>{{item.val}}</text>
    </view>
  </block>
</view>
</view>

```

代码清单 11-15 device 子页面的样式

device.wxss

```

.container {
  background-color: #eff4;
  width: 100%;
  height: 100%;
  flex-direction: column;
  display: flex;
  align-items: center;
  min-height: 100vh;
}

.category-item {
  width: 100%;
  margin: 20px 0;
  border-top: 1px solid #d9d9d9;
  border-bottom: 1px solid #d9d9d9;
  background-color: #fff;
}

.detail-item {
  display: flex;
  margin-left: 30px;
  border-bottom: 1px solid RGBA(217, 217, 217, .4);
}

```




```

height:85rpx;
align-items: center;
}
.detail-item:last-child{
border-bottom:none;
}
.detail-item text{
color:#7F8389;
font-size:24rpx;
flex:1;
}
.detail-item text:last-child{
color:#7F8389;
font-size:24rpx;
flex:1;
text-align: right;
padding-right: 20rpx;
}

```

MINA 框架所提供的获取系统信息的 API 是 `wx.getSystemInfo` 以及 `wx.getSystemInfoSync` 方法。前者是同步方法，后者是异步方法。选择同步还是异步，请根据自己的实际情况来考虑。以下是 `getSystemInfo` 方法可以获取的系统信息。

- `model`: 手机型号。
- `pixelRatio`: 设备像素比。
- `windowWidth`: 窗口宽度。
- `windowHeight`: 窗口高度。
- `language`: 微信设置的语言。
- `version`: 微信版本号。
- `system`: 操作系统版本。
- `platform`: 客户端平台。

`getSystemInfo` 的使用非常简单，在 `device.js` 文件中新增以下代码：

代码清单 11-16 获取系统信息

device.js

```

Page({
  data: {
    phoneInfo: [],
    softInfo: [],
    screenInfo: [],
  },
  onLoad: function () {
    var that = this;

```

```

wx.getSystemInfo({
  success: function (res) {
    that.setData({
      phoneInfo: [
        { key: "手机型号", val: res.model },
        { key: "手机语言", val: res.language }
      ],
      softInfo: [
        { key: "微信版本", val: res.version },
        { key: "操作系统版本", val: res.system },
        { key: "客户端平台", val: res.platform }
      ],
      screenInfo: [
        { key: "屏幕像素比", val: res.pixelRatio },
        { key: "屏幕尺寸", val: res.windowWidth + '×' + res.windowHeight }
      ]
    });
  }
});

```

保存并运行代码，点击【系统信息】后将打开如图 11-5 所示的 device（系统信息）页面。

要注意的是，在开发工具中以上信息多是模拟器模拟的参数。可以尝试在开发工具中调整当前机型，比如图 11-6 显示了当选择机型为 Nexus 6 时的系统信息。



图 11-5 系统信息

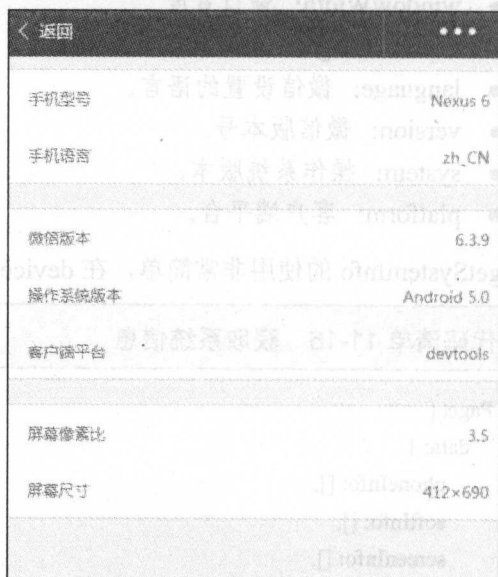


图 11-6 在模拟器中选择机型为 Nexus 6 时的系统信息

11.5 获取网络状态

MINA 框架提供了 `wx.getNetworkType` 作为获取当前网络状态的接口。获取网络状态是一个异步方法，方法回调函数中可以接收一个 `res` 参数，使用 `res.networkType` 可以获得当前移动设备的网络状态。网络状态的可能取值有 6 种：

- 2g
- 3g
- 4g
- Wifi
- none(140600 版本新增)
- unknown(140600 版本新增)

在 `setting.js` 文件中添加以下事件响应函数：

代码清单 11-17 获取网络状态信息

setting.js

```
//获取网络状态
showNetWork: function () {
    var that = this;
    wx.getNetworkType({
        success: function (res) {
            var networkType = res.networkType
            that.showModal('网络状态','您当前的网络: ' + networkType);
        }
    })
}
```

要注意的是，在开发工具中网络状态是由开发者自己设定的，如图 11-7 所示。

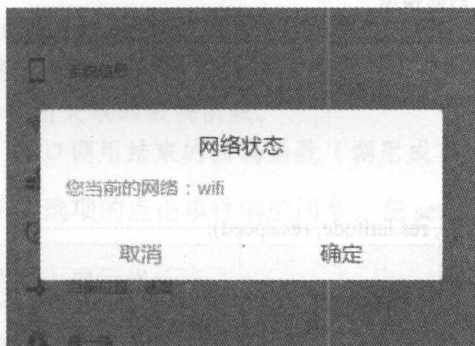


图 11-7 当前网络状态

11.6 获取当前位置信息与当前速度信息

MINA 提供 `wx.getLocation` 用来获取当前位置信息与当前速度信息。`wx.getLocation(OBJECT)` 的 `OBJECT` 有 4 个函数：

- type
- success
- fail
- complete

`success`、`fail` 和 `complete` 函数就不再赘述了，下面主要介绍 `type` 函数和 `success` 回调函数的返回结果。

`type` 指定一个返回地理位置信息的坐标系，默认是 WGS84 坐标系。除了 WGS84，还可以将 `type` 的值设为 GCJ02。WGS84 为国际标准 GPS 坐标，GCJ02 是中国测绘局订制的地理信息系统坐标系。如果开发者在获取当前位置后还想调用微信内置地图（调用 `wx.openLocation`）查看这个位置，那么请选用 GCJ02。如果你使用 WGS84 作为坐标系并调用微信内置地图，那么定位点的 `mark` 和实际位置是有偏差的。

`success` 回调函数接收一个 `object` 参数，参数中包含以下数据信息：

- latitude 纬度，浮点数，范围为-90~90，负数表示南纬。
- longitude 经度，浮点数，范围为-180~180，负数表示西经。
- speed 速度，浮点数，单位 m/s。
- accuracy 位置的精确度。

在 `setting.js` 中加入 `getLonLat` 和 `showLonLat` 函数。

代码清单 11-18 获取当前位置和速度信息

setting.js

```
//获取当前位置经纬度与当前速度
getLonLat: function (callback) {
  var that = this;
  wx.getLocation({
    type: 'gcj02',
    success: function (res) {
      callback(res.longitude, res.latitude, res.speed);
    }
  });
}
```

代码清单 11-19 显示当前位置和速度信息

setting.js

```
//显示当前位置坐标与当前速度
```




```

showLonLat:function(){
    var that=this;
    this.getLonLat(function(lon,lat,speed){
        var lonStr=lon>=0?'东经':'西经',
        latStr=lat>=0?'北纬':'南纬';
        lon=lon.toFixed(2);
        lat=lat.toFixed(2);
        lonStr+=lon;
        latStr+=lat;
        speed=(speed|| 0).toFixed(2);
        that.showModal('当前位置和速度','当前位置: '+lonStr+', '+latStr+'。速度: '+speed+'m/s');
    });
}
    
```

注意，第一次获取用户位置信息时是需要用户主动授权的，类似于获取用户基本信息，要注意处理用户拒绝授权的情况。以上代码编写完成后，点击“当前位置”和“速度”选项将弹出经纬度和速度信息。

11.7 使用微信内置地图查看位置信息

如果你有一个经纬度坐标点，可以使用微信内置的地图查看这个坐标点。注意坐标点的坐标系必须是 GCJ02，否则会出现实际位置和定位的 marker 点有偏差。

MINA 框架提供了一个 API `wx.openLocation(OBJECT)` 用于打开微信内置地图。`wx.openLocation` 的 object 参数对象属性如下：

- latitude Float 纬度，范围为-90~90，负数表示南纬。
- longitude Float 经度，范围为-180~180，负数表示西经。
- scale INT 缩放比例，范围 5~18，默认为 18。
- name String 位置名。
- address String 地址的详细说明。
- success Function 接口调用成功的回调函数。
- fail Function 接口调用失败的回调函数。
- complete Function 接口调用结束的回调函数（调用成功、失败都会执行）。

现在来编写【地图显示】选项的点击事件响应函数。在 `setting.js` 文件中加入以下代码：

代码清单 11-20 在地图上显示坐标点

setting.js

```

//在地图上显示当前位置
showMap: function () {
    this.getLonLat(function (lon, lat) {
        wx.openLocation({
    
```



```

latitude: lat,
longitude: lon,
scale: 15,
name: "海底捞",
address: "xx 街 xx 号",
fail: function () {
  wx.showToast({
    title: "地图打开失败",
    duration: 1000,
    icon: "cancel"
  });
}
});
}

```

点击【地图显示】后，将打开微信内置地图显示用户当前的位置。要注意，打开微信内置地图后通常将出现两个位置，一个红色的 marker 标记和一个实时移动的箭头。如果你传入的位置不是 GCJ02 坐标系，那么 marker 定位标记和实际位置是有偏差的。使用地图时最好在真机上预览效果，开发工具中的地图定位效果和功能都不是很完善。

11.8 监听罗盘数据制作一个简易指南针

小程序提供了一个监听罗盘数据的接口 `wx.onCompassChange`，监听频率是每秒 5 次。使用罗盘接口可以获取一个指示方向度数的返回值。我们可以使用罗盘监听函数编写一个简单的指南针。

首先，在 `setting.wxml` 的 `<view class=container>` 容器中增加一个 `modal` 组件。`modal` 组件的功能类似于我们经常使用的 `wx.showModal` 接口，区别在于一个是组件而另一个是 API 接口。它们的功能基本类似，这里使用 `modal` 组件是因为需要动态不间断地更改 `modal` 窗体的内容。

在 130400 版本中，`modal` 组件已从官方文档中移除，但还是可以在小程序中使用。本项目在这里简单介绍下 `modal` 组件的使用，其特点是可以非常方便地实现 `modal` 窗体 `content` 内容的动态更新，而 `wx.showModal` 无法实现动态更新 `content` 内容。如果在未来的某个版本中小程序不再支持 `modal`，指南针的 UI 将无法运行，请自行改写指南针 UI 部分。重要的不是 `modal` 组件，而是掌握罗盘接口使用方法。

代码清单 11-21 添加 modal 组件

setting.wxml

```

<modal title="指南针"
  confirm-text="确定"
  no-cancel hidden="{{compassHidden}}"
  bindconfirm="hideCompass"

```



```
confirm-color="#1F4BA5">
```

```
当前方向: 偏北{{compassVal}}°
```

```
</modal>
```

compassHidden 控制 modal 组件的显示和隐藏,默认 compass modal 是隐藏的。bindconfirm 属性指定当点击“确定”时需要执行的事件响应函数。下面我们在 setting.js 中编写指南针的逻辑代码,在 setting.js 中指定初始化 compassHidden 变量、编写 hideCompass 事件响应函数以及调用 wx.onCompassChange 接口。

代码清单 11-22 初始化变量值

setting.js

```
data: {
  // ..... 省略若干行代码

  compassVal: 0,
  compassHidden:true
}
```

compassVal 作为指南针的数据绑定量,默认值是 0。

代码清单 11-23 显示罗盘

setting.js

```
//显示罗盘
showCompass: function () {
  var that = this;
  this.setData({
    compassHidden: false
  })
  wx.onCompassChange(function (res) {
    if (!that.data.compassHidden) {
      that.setData({ compassVal: res.direction.toFixed(2) });
    }
  });
}
```

在 1 秒钟内,MINA 框架将连续调用 5 次 wx.onCompassChange 内的回调函数。这样我们就可以不断更新指南针的方向度数。

最后,加上一个隐藏指南针的函数即可。

代码清单 11-24 隐藏罗盘

setting.js

```
//隐藏罗盘
hideCompass: function () {
  this.setData({
```



```
compassHidden: true
  })
}
```

如图 11-8 所示为点击【指南针】选项后的真机效果图。

要注意的是，罗盘监听接口只有在真机中才能被正常调用，在模拟器中是无法正常运行指南针功能的。

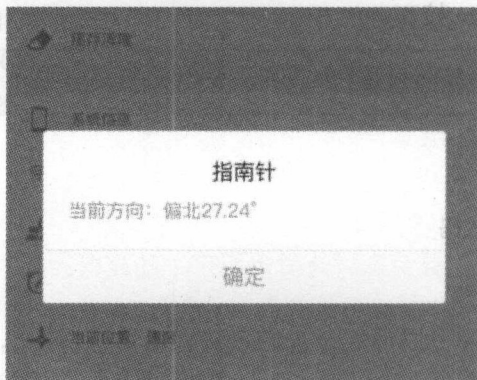


图 11-8 指南针的真机效果图

11.9 在小程序中实现摇一摇

摇一摇一直是微信经典的小功能。本节将实现一个小程序版本的摇一摇功能。具体效果为：如果摇一摇成功，那么播放一段音频，并将摇一摇次数累加 1。注意，此功能需要真机支持，在模拟器中无法正常运行。当然，次数加 1 只是 1 个示例，你可以实现自己的业务逻辑。

摇一摇的具体实现原理为：利用手机的重力感应机制监听手机摇晃幅度。如果手机摇晃幅度超过一定偏移量就认为摇一摇成功，可以执行具体业务；如果手机摇晃幅度不大就认为不是摇一摇。

MINA 框架提供的重力感应监听函数为 `wx.onAccelerometerChange(CALLBACK)`。同罗盘监听函数一样，重力感应监听函数统一为 1 秒钟响应 5 次。CALLBACK 回调函数中将可以获得重力感应在 *x*、*y*、*z* 轴上的值。

同 11.8 小节的指南针功能类似，首先要在 `setting.wx.xml` 中添加一个 `modal` 组件用来显示摇一摇的次数。

代码清单 11-25 添加摇一摇计数面板

setting.wx.xml

```
<modal title="摇一摇"
  confirm-text="确定"
  no-cancel hidden="{{shakeInfo.gravityModalHidden}}"
  bindconfirm="gravityModalConfirm"
  confirm-color="#1F4BA5">
```



当前摇一摇次数: {{shakeInfo.num}}

</modal>

接着, 在 setting.js 中初始化摇一摇所需要的状态变量。在 setting.js 的 data 变量中添加以下代码:

代码清单 11-26 添加 shakeInfo 和 shakeData 变量

setting.js

```
data: {
    //.....省略若干代码
    shakeInfo: {
        gravityModalHidden: true,
        num: 0,
        enabled: false
    },
    shakeData: {
        x: 0,
        y: 0,
        z: 0
    }
}
```

再编写摇一摇的具体实现代码。在 setting.js 中增加一个 shake 函数, 代码如下:

代码清单 11-27 摇一摇的具体实现

setting.js

```
//摇一摇
shake: function () {
    var that = this;
    //启用摇一摇
    this.gravityModalConfirm(true);

    wx.onAccelerometerChange(function (res) {
        //摇一摇核心代码, 判断手机晃动幅度
        var x = res.x.toFixed(4),
            y = res.y.toFixed(4),
            z = res.z.toFixed(4);
        var flagX = that.getDelFlag(x, that.data.shakeData.x),
            flagY = that.getDelFlag(y, that.data.shakeData.y),
            flagZ = that.getDelFlag(z, that.data.shakeData.z);

        that.data.shakeData = {
            x: res.x.toFixed(4),
            y: res.y.toFixed(4),
            z: res.z.toFixed(4)
        }
    });
}
```



```

    });
    if (flagX && flagY || flagX && flagZ || flagY && flagZ) {
        // 如果摇一摇幅度足够大，就认为摇一摇成功
        if (that.data.shakeInfo.enabled) {
            that.data.shakeInfo.enabled = false;
            that.playShakeAudio();
        }
    }
    });
}

```

以上代码中调用了 3 个方法：gravityModalConfirm 方法用来显示、隐藏摇一摇技术面板以及开启和停用摇一摇功能；getDelFlag 方法用来计算摇一摇的偏移量；playShakeAudio 方法是在摇一摇成功后播放音频并计数。

代码清单 11-28 开启或关闭摇一摇

setting.js

```

//启用或停用摇一摇功能
gravityModalConfirm: function (flag) {
    if (flag !== true) {
        flag = false;
    }
    var that = this;
    this.setData({
        shakeInfo: {
            gravityModalHidden: !that.data.shakeInfo.gravityModalHidden,
            num: 0,
            enabled: flag
        }
    })
}

```

代码清单 11-29 计算偏移量

setting.js

```

//计算摇一摇的偏移量
getDelFlag: function (val1, val2) {
    return (Math.abs(val1 - val2) >= 1);
}

```

代码清单 11-30 摇一摇成功后播放声音

setting.js

```

// 摇一摇成功后播放声音并累加摇一摇次数
playShakeAudio: function () {
    var that = this;

```



```

wx.playBackgroundAudio({
  dataUrl: 'http://7xqnxu.com1.z0.glb.clouddn.com/wx_app_shake.mp3',
  title: "",
  coverImgUrl: ""
});
wx.onBackgroundAudioStop(function () {
  that.data.shakeInfo.num++;
  that.setData({
    shakeInfo: {
      num: that.data.shakeInfo.num,
      enabled: true,
      gravityModalHidden: false
    }
  });
});
}

```

以上代码是摇一摇的功能代码。晃动手机后，首先响起“咔嚓”声，随后计数面板的数字将增加 1。请注意，声音文件位于我们提供的一个外网文件，你可以替换成自己的声音文件。

11.10 扫 码

扫码不仅是指扫描常见的二维码，而且支持扫描一维码。扫码的功能对于主打线下的小程序非常重要，开发者可以充分利用扫码功能做出各种个性化、场景化的业务功能。

MINA 框架提供了 `wx.scanCode` 用于扫描二维码，其回调函数参数对象包含以下 4 个返回值：

- `result` 扫码的内容。
- `scanType` 扫码的类型。
- `charSet` 扫码的字符集。
- `path` 当所扫的码为当前小程序的合法二维码时，会返回此字段，内容为二维码携带的 `path`。

下面我们来实现【二维码】选项。在 `setting.js` 中添加 `scanQRCode` 函数。

代码清单 11-31 实现扫码功能

setting.js

```
//扫码
```

```

scanQRCode: function () {
  var that = this;
  wx.scanCode({
    success: function (res) {
      console.log(res)
    }
  })
}

```



```

    that.showModal('扫描二维码/条形码', res.result, false);
  },
  fail: function(res) {
    that.showModal('扫描二维码/条形码', "扫描失败，请重试", false);
  }
}
})
}

```

完成以上代码后，如果在模拟器中点击“二维码”，就会打开“操作系统文件选择”对话框，让你选择一种二维码/条形码图片；如果是在真机上，就会打开相机让你扫描。

在模拟器中为什么不可以调用 PC 上的摄像头扫码呢？因为同手机端直接调用摄像头扫码不同，在 PC 上调用摄像头扫码完成调试是很低效的行为，所以在开发工具上调用二维码扫码 API 后，开发者可以选择一张本地图片进行后续的逻辑调试，而不是真正的启用摄像头扫码，流程有所不同，但是接口的输入和输出是一致的。

开发者可扫描如图 11-9 所示的图片查看扫码效果。扫描上面这张二维码后，将弹出一个对话框显示唐代诗人元稹的《离思五首·其四》，如图 11-10 所示。



图 11-9 示例二维码

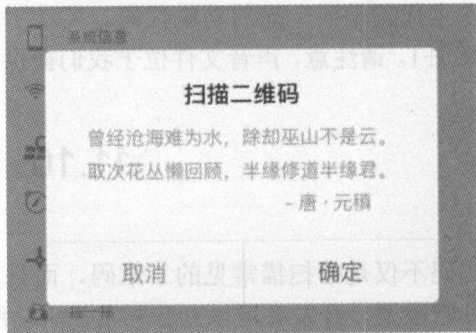


图 11-10 二维码扫描效果

再来看除了 `res.result` 外的其他几个返回结果值。图 11-11 所示是 `console.log(res)` 的结果。

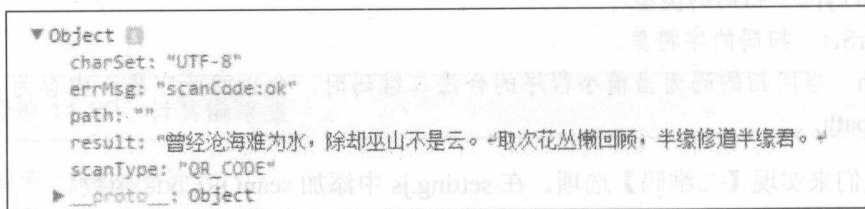


图 11-11 wx.scanCode 的全部返回值

这里要特别说明的是 `path` 这个返回结果。通常情况下，自定义二维码时这个 `path` 的结果都是空字符串。官方文档中是这么描述 `path` 的：当所扫的码为当前小程序的合法二维码时会返回此字段，内容为二维码携带的 `path`。官方的描述并不是非常清楚，基本上是看不懂的。下面我们尝试对这个描述做一些解释。

除了普通的二维码（就是自己生成的带有自定义内容的二维码，比如上面包含元稹诗词的二维码）外，还有一类二维码是小程序的专用二维码，用于扫描后直接打开小程序的某个页面。注意，这类二维码可以直达小程序内部的某个页面，只有扫描这类二维码时 `path` 才是有值的。

在开发工具中也可以找到一个这样的二维码。请按照以下步骤获取二维码：点击开发工具中的【项目】。在【项目】面板的中间有一个绿色的【预览】，点击【预览】右侧的下拉小箭头，选择自定义预览，会出现如图 11-12 所示的设置启动页面和启动页面参数面板。

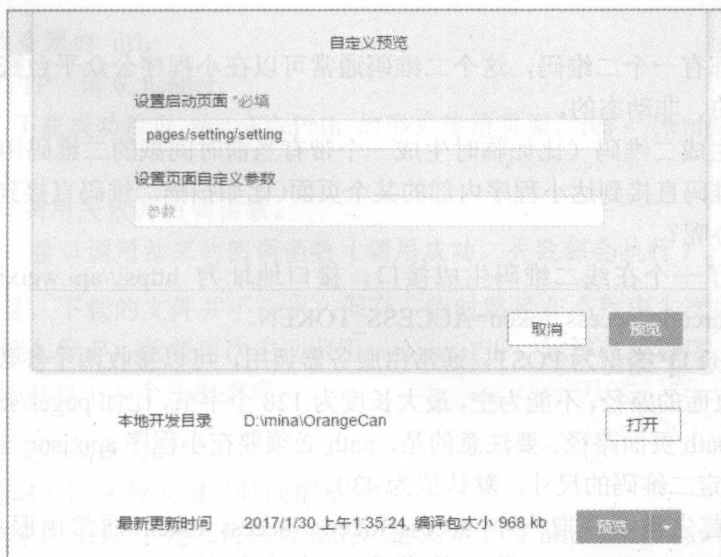


图 11-12 自定义预览

设置启动页面路径后，会弹出一个二维码，使用刚刚编写好的【二维码】功能扫描这个二维码（在真机上），或者将此二维码保存为图片并使用【二维码】功能选择这个二维码（使用开发工具）。此时打印出来的 `res` 的 `path` 将显示为我们设置的启动页面路径：“`pages/setting/setting?`”，如图 11-13 所示。

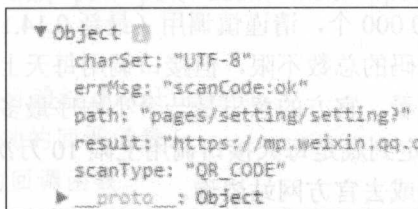


图 11-13 带 `path` 的二维码扫描结果

这类二维码用于直接打开小程序内部页面的二维码，同我们自定义的二维码略有不同。

这里顺便介绍一下在【项目】中自定义预览的功能。在 6.4 小节中，我们介绍过如何在开发工具中指定小程序的启动页面，这个功能让我们可以很方便地在开发工具中直接打开某个小程序页面，而无需每次启动小程序时都从 `welcome` 页面进入小程序。

【调试】中的【设置启动页面】只对开发工具有效，如果想在真机上直接打开小程序的某个页面，就不能使用【调试】中的【设置启动页面】功能。但我们可以使用【项目】下的【自定义预览】功能，使用该功能生成一个直达页面的二维码，扫描二维码后即可直达指定路径的小程序页面。

11.11 获取小程序页面二维码

每个小程序都有一个二维码，这个二维码通常可以在小程序公众平台账号中获得。但这类二维码是固定的、非动态的。

如果想动态生成二维码（比如临时生成一个带有当前时间戳的二维码用于打开签到），还想能够扫描二维码直接到达小程序内部的某个页面（比如扫描二维码直接到达某个商品的详情页面）该怎么办呢？

小程序提供了一个在线二维码生成接口，接口地址为 https://api.weixin.qq.com/cgi-bin/wxaapp/createwxaqrcode?access_token=ACCESS_TOKEN。

这个接口的 HTTP 类型为 POST，通常由服务器调用，可以接收两个参数：path 和 width。path 指定小程序页面的路径，不能为空，最大长度为 128 个字节，比如 pages/setting/setting?id=3 就是一个合法的 path 页面路径。要注意的是，path 必须要在小程序 app.json 文件的 pages 数组下注册。width 指定二维码的尺寸，默认值为 430。

调用此接口首先需要获取一个 access_token。access_token 通常由服务器获取。关于 access_token 以及 access_token 具体的获取方法请参考 <https://mp.weixin.qq.com/wiki?id=mp1421140183>。

在后续微信模板消息的章节中，我们将在 php 代码中演示如何获取 access_token。此外，调用二维码生成接口还要注意以下 4 点：

- （1）通过该接口仅能生成已发布的小程序的二维码。
- （2）可以在开发者工具预览时生成开发版的带参二维码。
- （3）带参二维码只有 100 000 个，请谨慎调用（最新 0.14.140900 版本）。但在老版本说明中，官方描述是：生成二维码的总数不限，但接口调用每天上限 10 000 次（130400 版本中的文档说明）。从字面意思来看，官方的意思是一个小程序最多只能调用 10 万次此接口。此接口难以测试，目前还不能确定到底是每天接口调用上限 10 万次还是总次数 10 万次。开发者开发时，可再次查看官方文档或去官方网站咨询。

- （4）POST 参数需要转成 json 字符串，不支持 form 表单提交。

如果你正在制作线下小程序，比如点餐服务，每个餐台都有对应的餐台号，你想让顾客扫描二维码打开小程序且附带餐台号，那么应该使用本小节所描述的方法生成一组含有不同餐台号的二维码，将其打印出来给顾客扫码使用。



11.12 下载并预览 pdf、word 等多种类型文档

可以从网络上下载文件并在本地预览。小程序提供了 `wx.downloadFile(OBJECT)` 方法用于下载文件资源到本地，该方法将发起一个 `http` 的 `GET` 请求并返回文件在本地的临时路径。`wx.downloadFile(OBJECT)` 的 `OBJECT` 参数说明如下：

- `url` 下载资源的 `url`。
- `header` `HTTP` 请求 `header`。
- `success` 下载成功后以 `tempFilePath` 的形式传给页面，`res = {tempFilePath: '文件的临时路径'}`。
- `fail` 接口调用失败的回调函数。
- `complete` 接口调用结束的回调函数（调用成功、失败都会执行）。

需要注意的是，下载的文件并不会永久保存，临时路径在小程序本次启动期间可以正常使用，如果需要持久保存，就要再次主动调用 `wx.saveFile`，这样在小程序下次启动时才能访问得到。此外，还有以下 4 个注意事项：

- (1) 最大并发限制是 10 个。
- (2) 默认超时时间和最大超时时间都是 60s。
- (3) 网络请求的 `referer` 是不可以设置的，格式固定为 `https://servicewechat.com/{appid}/{version}/page-frame.html`。其中，`{appid}` 为小程序的 `appid`，`{version}` 为小程序的版本号，版本号为 0 表示为开发版。

(4) 同 `wx.request` 类似，`wx.downloadFile` 的 `URL` 地址同样需要加入小程序的可信域名列表。

同时，小程序还提供了一个 `wx.openDocument(OBJECT)` 方法用于打开一个新页面预览文档，支持的文件类型有 `doc`、`xls`、`ppt`、`pdf`、`docx`、`xlsx`、`pptx`。`wx.openDocument(OBJECT)` 的 `OBJECT` 参数说明如下：

- `filePath` 文件路径，可通过 `downFile` 获得。
- `success` 接口调用成功的回调函数。
- `fail` 接口调用失败的回调函数。
- `complete` 接口调用结束的回调函数（调用成功、失败都会执行）。

要注意的是，预览文件只在真机上有效，在开发工具中是没有效果的。

下面实现下载 `pdf`、`word` 文档功能。首先，在 `app.json` 的 `pages` 数组下新增一个 `download` 页面。

代码清单 11-32 新增 download 页面

app.json

```

"pages": [
  "pages/welcome/welcome",
  "pages/post/post",
  "pages/post/post-detail/post-detail",
  "pages/post/post-comment/post-comment",
  "pages/movie/movie",
  "pages/movie/more-movie/more-movie",
  "pages/movie/movie-detail/movie-detail",
  "pages/setting/setting",
  "pages/setting/device/device",
  "pages/setting/document/download/download"
]

```

黑色加粗部分为新增页面。点击【下载 pdf、word】后，将跳转到 download 页面。编写【下载 pdf、word】的事件响应函数 downloadDocumentList。

代码清单 11-33 跳转到 download 页面

setting.js

```

downloadDocumentList: function () {
  wx.navigateTo({
    url: 'pages/setting/document/download/download'
  });
}

```

接着，编写 download 页面的骨架和样式。在 download.wxml 文件中新增以下代码：

代码清单 11-34 download 页面的骨架代码

download.wxml

```

<view class="container">
  <view class="file-type-head">
    <text>文件类型</text>
  </view>
  <view class="category-item">
    <block wx:for="{{fileTypeList}}">
      <view class="detail-item" catchtap="downloadFile"
        data-type="{{item.type}}">
        <image src="{{item.iconurl}}"></image>
        <text>{{item.title}}</text>
        <view class="detail-item-btn"></view>
      </view>
    </block>
  </view>
</view>

```




```

</block>
</view>
</view>

```

接着，在 download.wxss 文件中新增以下代码：

代码清单 11-35 download 页面的样式代码

download.wxss

```

.file-type-head {
  display: flex;
  align-items: center;
  height: 70rpx;
  width: 100%;
  text-indent: 30rpx;
}

.file-type-head text {
  font-size: 32rpx;
}

.category-item {
  margin-top: 0;
}

.container {
  background-color: #efff4;
  width: 100%;
  height: 100%;
  flex-direction: column;
  display: flex;
  align-items: center;
  min-height: 100vh;
}

.category-item {
  width: 100%;
  margin: 20rpx 0;
  border-top: 1rpx solid #d9d9d9;
  border-bottom: 1rpx solid #d9d9d9;
  background-color: #fff;
}

.detail-item {
  display: flex;

```

```
margin-left: 30rpx;  
border-bottom: 1px solid RGBA(217, 217, 217, 0.4);  
height: 85rpx;  
align-items: center;  
}
```

```
.detail-item:last-child {  
border-bottom: none;  
}
```

```
.detail-item image {  
height: 40rpx;  
width: 40rpx;  
}
```

```
.detail-item text {  
color: #7f8389;  
font-size: 24rpx;  
flex: 1;  
margin-left: 30rpx;  
}
```

```
.detail-item .detail-item-btn {  
width: 50rpx;  
color: #d9d9d9;  
height: 40rpx;  
margin-right: 20rpx;  
text-align: center;  
}
```

```
.detail-item .detail-item-btn::after {  
display: inline-block;  
content: " ";  
width: 16rpx;  
height: 16rpx;  
color: #d9d9d9;  
margin-top: 8rpx;  
border: 3rpx #d9d9d9 solid;  
border-top-color: transparent;  
border-left-color: transparent;  
transform: rotate(-45deg);  
}
```



随后，我们需要在 `download.js` 文件中配置 `data` 变量下的文件类型数组，配置完成后 `download` 页面才能正常显示。在 `download.js` 文件中添加以下代码：

代码清单 11-36 配置文件类型

`download.js`

```
Page({
  data: {
    fileTypeList: [
      { type: 'pdf', iconurl: '/images/icon/wx_app_pdf.png', title: 'pdf 类型' },
      { type: 'word', iconurl: '/images/icon/wx_app_word.png', title: 'word 类型' },
      { type: 'excel', iconurl: '/images/icon/wx_app_exl.png', title: 'excel 类型' },
      { type: 'ppt', iconurl: '/images/icon/wx_app_ppt.png', title: 'ppt 类型' }
    ],
  },
});
```

保存代码后，`download` 页面将呈现如图 11-14 所示的 UI 效果。

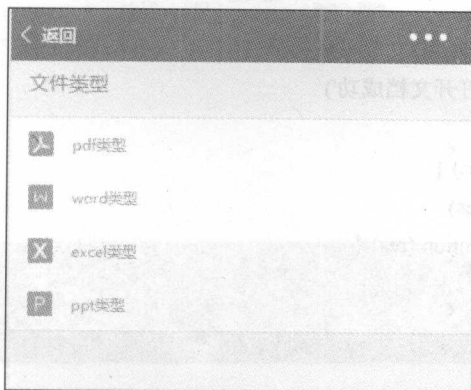


图 11-14 `download` 页面效果

最后，编写具体的下载文件和预览文档的业务逻辑代码。在 `document.js` 文件中添加事件响应函数 `downloadFile`。

代码清单 11-37 下载并预览文档

`document.js`

```
downloadFile: function (event) {
  var type = event.currentTarget.dataset.type,
    url = 'https://coding.net/u/airbreak/p/wx_app_files/git/raw/master/top10.';
  switch (type) {
    case "pdf":
      url += 'pdf';
      break;
```

```

    case "word":
        url += 'docx';
        break;
    case "excel":
        url += 'xlsx';
        break;
    default:
        url += 'pptx';
        break;
}
wx.downloadFile({
    url: url,
    success: function (res) {
        var filePath = res.tempFilePath;
        console.log(filePath);
        wx.openDocument({
            filePath: filePath,
            success: function (res) {
                console.log('打开文档成功')
            },
            fail: function (res) {
                console.log(res)
            }, complete: function (res) {
                console.log(res);
            }
        })
    },
    fail: function () {
        console.log('下载失败');
    }
})
}

```

首先，调用 `wx.downloadFile` 下载文件，文件下载成功后（`success` 回调函数内部）再调用 `wx.openDocument` 打开并预览下载文档。代码中的文档文件 `url` 是我们提供的一个示例地址，可以被访问，也可以配置自己的文档地址。

如果想在真机上测试本示例，记得将示例中的文档 `url` 域名添加到小程序公众账号的可信域名列表中，否则无法下载和预览这些文档文件。

图 11-15 所示是在真机上打开 pdf 文档的示意图。





图 11-15 在真机上预览 pdf 文档的效果图

Word、Excel、PPT 类型的文档也可以正常预览。

第 12 章

开放接口

本章我们将学习微信开放接口。毫不夸张地说，微信的开放接口是微信类产品最有价值的一部分。没有微信开放接口，小程序的价值将大打折扣。即使你不是一个微信开发者或者你的小程序无须使用微信开放接口，但理解微信开放接口的设计原理对提升架构方面的知识也很有好处。

关于开放接口这块儿，微信官方文档写得并不够详尽，本章将详细地介绍用户登录、用户信息校验、解析用户加密数据获取 `openId`、模板消息以及最重要的微信支付。

12.1 准备工作

调用微信小程序的开放接口要求开发者必须拥有小程序账号。拥有小程序账号的开发者拥有一个 `appId`，如果你的 Orange Can 现在没有 `appId`，请将 `appId` 加入 Orange Can 项目中。目前小程序开发工具没有提供加入 `appId` 的快捷方式，所以请开发者新建一个带有 `appId` 的项目并将目录指向 Orange Can 的文件目录。

本章内容需要使用小程序的 `appSecret`，小程序的 `appId` 和 `appSecret` 均可以在小程序账户的【设置】→【开发设置】→【开发者 ID】中获得。

本章要实现的功能还需要调用服务器接口。本书中的服务器代码均采用 PHP 编写，并部署在本地的 Apache 服务器中。本章会给出所有 PHP 示例代码，你也可以下载我们提供的 PHP 示例代码，并部署在本地或网络服务器上。以下 url 地址是用户登录接口的示例地址：

`http://localhost:8080/wxopen/wxlogin.php`

注意，如果在项目中填写了 `appId`，且服务器在本地，那么调用本地 http 接口时一定要勾选【项目】中的【开发环境不校验请求域名以及 TLS 版本】，否则无法访问服务器。

为了简化代码，本章的代码（小程序和 PHP 中的代码）都没有做过多异常处理，只能作为演示功能的示例代码。Orange Can 并不是真实的项目，请各位开发者以理解功能原理为主要目的。如果要开发自己的真实项目，请考虑各种调用失败的情况并加强安全性。

注意，本章的代码依然用于 `setting` 页面，每个功能都将作为 `setting` 页面的一个子项，本章 `setting` 相关的子项配置已经在 11.1 节的代码清单 11-5 中添加完毕。同时，本章大多数代码的运行结果与之前的章节不同，之前的章节我们都会使用 UI 效果展示运行结果，而本章大部分示例代码都将以 `console.log` 的方式输出到控制台中，请开发者自行在“Console”面板中查看代码的运行结果。

12.2 用户登录

用户登录是小程序中获取用户加密信息、使用模板消息、客服消息、微信支付等接口的先决条件。只有用户在小程序中登录了，我们才能获取用户在当前小程序中的 `openId` 以及本次登录会话的 `session_key`。`openId` 是使用一系列微信小程序开放接口的必要参数。关于 `openId` 的相关内容我们在 11.2 节中已经具体讲解过，这里就不再赘述。

首先，我们要了解什么是微信小程序登录，有什么作用？

微信小程序登录是为了让开发者的服务器获取用户的 `openId` 以及 `session_key` 的令牌。

请不要将微信小程序的用户登录理解为传统意义上的登录。虽然从大的方向上讲，登录都是为了确认用户的身份、拿到用户的唯一身份标识，但是微信小程序的登录流程和传统的账号密码流程不太一样。当你进入小程序后，不需要输入任何账号密码，开发者服务器就可以获取你的 `openId` 标识。微信小程序的登录流程远比传统的账号密码登录要复杂。



微信小程序提供了 `wx.login(OBJECT)` 方法用于用户登录。`wx.login` 方法的主要目的是拿到用户的 `openId` 和用户本次登录的 `session_key`。

`openId` 是用户对于当前小程序的身份标识，类似于我们自己产品里的用户 `id` 号，只不过在微信里有自己的用户账户体系，你可以使用这个 `openId` 作为用户的身份标识。当然，也可以建立一套自己的用户标识，不过当你需要调用微信开放接口（比如支付、发送模板消息）时，还是必须知道用户的 `openId`。

`session_key` 是本次用户登录的会话密钥，通常用来对用户的通信数据进行加解密。

到目前为止，如果对于 `openId` 和 `session_key` 的概念还很模糊，不知道这两者的用处，没有关系，在后面的小节中我们将陆续讲解和使用这两个参数。在本小节中，首先要在开发者服务器拿到 `openId` 和 `session_key`。微信开放 API 中有很多概念不是那么好理解，必须用实例代码进行演示讲解。

图 12-1 所示为获取 `openId` 和 `session_key` 的流程图。这张图非常重要，可能你现在还看不明白，在阅读本小节后面的内容时可时时回顾一下。

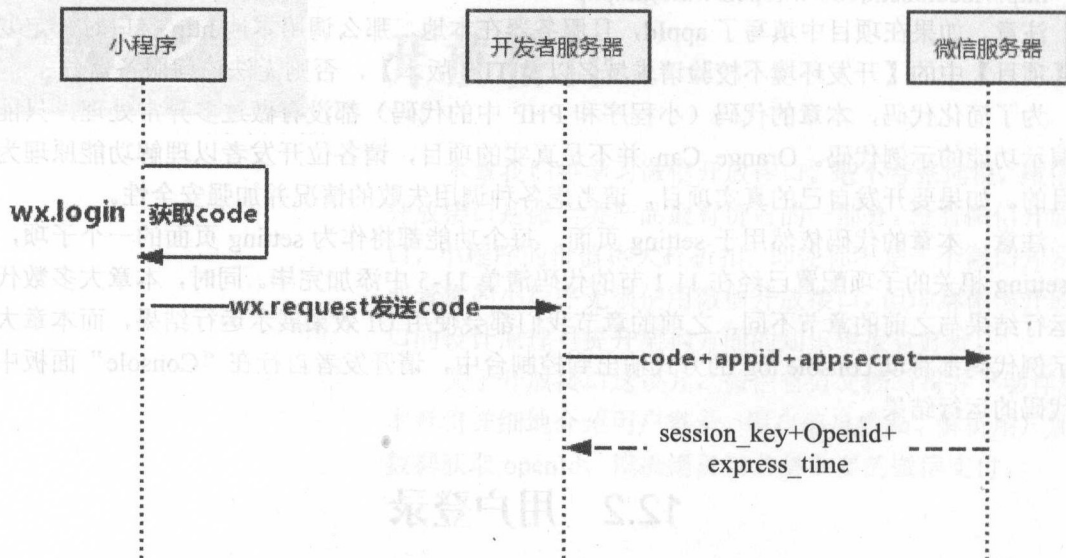


图 12-1 登录并获取 `session_key` 和 `openId` 的流程图

要获取 `session_key` 和 `openId`，首先需要在小程序中调用 `wx.login`，并获取 `code`；随后将 `code` 发送到开发者服务器，并同 `appid` 和 `appsecret` 一起发送到微信服务器，微信服务器会返回我们需要的 `session_key` 和 `openId`。

下面编写具体的代码，实现上述流程并最终拿到 `session_key` 和 `openId`。

首先，在 `/pages/setting/open-api` 下新建 `login` 目录及 `login` 页面的 4 个文件用来演示用户登录的代码和效果。

在 `app.json` 文件的 `pages` 数组下新增 `login` 页面。



代码清单 12-1 新增 login 页面

app.json

```
"pages": [
  //.....省略其他页面路径
  "pages/setting/open-api/login/login"
]
```

接着，在 login.wxml 中增加一个 button 按钮。

代码清单 12-2 添加 button 按钮

login.wxml

```
<button type="primary" bindtap="onTap">用户登录</button>
```

在 button 按钮上注册了一个时间 onTap，后面将在 onTap 函数中调用 wx.login(OBJECT) 方法。

在 setting.js 文件中添加 login 事件响应函数，点击 setting 页面的“用户登录”选项后，将跳转到 login 页面。

代码清单 12-3 跳转到 login 页面

setting.js

```
login: function () {
  wx.navigateTo({
    url: '/pages/setting/open-api/login/login'
  });
}
```

做完以上准备工作后，点击 setting 页面的“用户登录”子项，将跳转到如图 12-2 所示的页面。

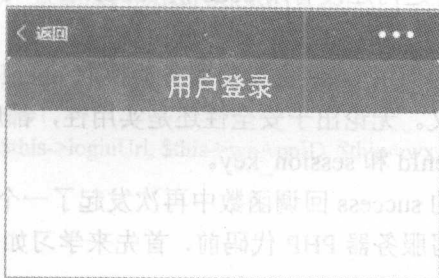


图 12-2 login 页面

下面编写 login 页面的按钮响应时间 onTap。在 login.js 中添加 onTap 事件响应函数。

代码清单 12-4 添加 onTap 事件响应函数

login.js

```
Page({
  data: {},
  onTap: function () {
```

```

wx.login({
  success: function (res) {
    console.log('code:'+res.code);
    wx.request({
      url: "http://localhost:8080/wxopen/wxlogin.php",
      data: {
        code: res.code
      },
      success: function (res) {
        console.log(res.data);
      }
    })
  }
})

```

用户在 login 页面点击【用户登录】后，将执行 onTap 函数。我们在 onTap 函数中调用了 wx.login 方法，这个方法没有参数，只有回调函数。success 回调函数是当微信服务器成功返回结果时调用的函数。

先来看 success 回调函数返回参数的参数说明。

- errMsg 错误消息。
- code 开发者需要将 code 发送到开发者服务器后台，使用 code 换取 session_key api，将 code 换成 openId 和 session_key。

重点是返回值 code。code 是一把钥匙，是得到 openId 和 session_key 的关键。code 有效期只有 5 分钟，如果在 5 分钟之内还没有用 code 换取 openId 和 session_key，那么就不能再使用了。

可不可以直接在小程序内部用 code 去微信服务器换取 openId 和 session_key 呢？理论上是可以的，但这完全没有意义。无论出于安全性还是实用性，都应该将 code 发送到开发者服务器由开发者服务器获取 openId 和 session_key。

所以，我们在 wx.login 的 success 回调函数中再次发起了一个 wx.request 请求，将 code 发送到了本地服务器中。在编写服务器 PHP 代码前，首先来学习如何使用 code 调用微信服务器获取 openId 和 session_key。

微信提供了一个 https 接口用于 code 换取 openId 以及 session_key。接口地址如下：
https://api.weixin.qq.com/sns/jscode2session?appid=APPID&secret=SECRET&js_code=JSCODE&grant_type=authorization_code

需要以下 4 个参数才能正确调用该接口：

- appid 小程序唯一标识。
- secret 小程序的 app secret。
- js_code 登录时获取的 code。



- grant_type 填写为 authorization_code。

appid 和 secret 均来自于微信小程序账号，是 2 个固定的字符串；js_code 是我们在小程序客户端调用 wx.login 时返回的 code；grant_type 固定为 authorization_code 即可。

下面编写服务端代码。先给出服务器 PHP 代码的目录示意图，如图 12-3 所示。

首先，编写一个 PHP 的类 WXLogin，位于 wxLoginClass.php 中，wxLoginClass.php 文件位于根目录的 class 目录下。WXLogin 主要负责用 code 换取 openId 和 session_key。

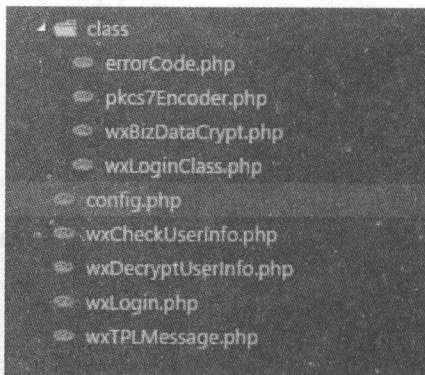


图 12-3 服务器 PHP 代码目录示意图

代码清单 12-5 WXLogin Class

wxLoginClass.php

```
<?php
```

```
class WXLogin{
```

```
    private $loginUrl;
```

```
    private $wxAppID;
```

```
    private $wxAppSecret;
```

```
    function WXLogin(){
```

```
        $config = include('./config.php');
```

```
        $this->loginUrl = $config['LOGINURL'];
```

```
        $this->wxAppID = $config['WXAPPID'];
```

```
        $this->wxAppSecret = $config['WXAPPSECRET'];
```

```
    }
```

```
    public function login($code){
```

```
        $loginUrl = sprintf($this->loginUrl, $this->wxAppID, $this->wxAppSecret,$code);
```

```
        $ch = curl_init();
```

```
        $timeout = 10;
```

```
        try{
```

```
            curl_setopt ($ch, CURLOPT_URL, $loginUrl);
```

```
            curl_setopt ($ch, CURLOPT_RETURNTRANSFER, 1);
```

```
            curl_setopt ($ch, CURLOPT_CONNECTTIMEOUT, $timeout);
```

```
            curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, false);
```

```
            $res = curl_exec($ch);
```

```
            curl_close($ch);
```

```
            $resArray = json_decode($res,true);
```

```
            return $resArray;
```

```
        }
```

```

        catch (Exception $e){
            return $e;
        }
    }
}

```

以上代码需要 PHP 开启 curl 支持。在代码中，我们 include 了一个 config.php 的配置文件，config.php 文件位于根目录下，该配置文件主要用于存放小程序的 appid 以及 appsecret。

代码清单 12-6 编写配置文件 config.php

config.php

```

<?php
return array(
    'WXAPPID' => 'your appid', //请替换成自己的 appid
    'WXAPPSECRET' => 'your appsecret', //请替换成自己的 secret
    'LOGINURL' => "https://api.weixin.qq.com/sns/jscode2session?".
        "appid=%s&secret=%s&js_code=%s&grant_type=authorization_code",
);

```

请开发者自行将 appid 和 appsecret 替换为自己的 appid 和 secret。看到 config.php 中的代码后，开发者可能就会明白为什么不能在小程序客户端中换取 openId 和 session_key。因为调用换取接口需要敏感数据 appid 和 appsecret，特别是 secret，一定不可以放在客户端中。

有了 config.php 和 wxLoginClass.php，就可以编写 wxlogin.php 接口文件了，该文件位于根目录下。

代码清单 12-7 向客户端提供换取 openId 接口

wxlogin.php

```

<?php
# 用户登录示例代码
# 用户登录主要是去微信服务器获取 session_key 和用户的 openId

include_once('./class/wxLoginClass.php');

$code = $_GET['code'];
$login = new WXLogin();
echo json_encode($login->login($code));

```

接口代码很简单，接收小程序客户端传来的 code 值，并通过 WXLogin 类调用微信服务器，最终拿到 openId 和 session_key。

要特别注意，以上代码为了演示效果，所以将拿到的 openId 以及 session_key 返回到了客户端。但在真实的项目中，将 session_key 和 openId 返回客户端是极其危险的，也完全没有必要。因为需要使用 session_key 和 openId 的场景都会被放置在服务器进行，所以将这两个参数返回小程序中没有任何意义，反而会增加数据泄露的风险。

以上完成了用户登录的全部服务器端代码。



运行客户端小程序，点击 login 页面的“用户登录”按钮后，console 的显示结果如图 12-4 所示结果。

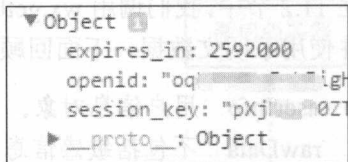


图 12-4 由 code 换取的 openid 和 session_key

注意，除了 openid 和 session_key，服务器还返回了一个 expires_in 参数。这个参数的数值官方并没有在文档中提到，笔者猜测这个数值是 session_key 的失效时间戳，单位推测为秒，换算为天是 30 天。由于失效期太长，目前没有验证过，所以以上结论仅是推测（在开发工具目前的最新版本中，这个过期时间已被更改为 7200，单位未知，推测单位是秒，也就是 2 小时）。

session_key 肯定是有失效期的。要注意的是，在 session_key 的有效期内，开发者最好不要重复调用 wx.login 接口、不断用 code 换取 session_key，而应该将 session_key 保存在服务器中。等到 session_key 失效后，再重新获取新的 session_key。

那么如何知道 session_key 是否已经过期呢？小程序提供了一个 wx.checkSession(OBJECT) 用来校验 session_key 是否过期。只有在 session_key 确实过期后，才会再次调用 wx.login。当然，Orange Can 是一个示例项目，并不会在服务器记录 session_key，所以每次操作都会重新调用 wx.login 接口换取新的 session_key。wx.checkSession 的使用非常简单，下面直接给出示例代码。

代码清单 12-8 检查 session 是否过期

```
wx.checkSession({
  success: function() {
    //登录状态未过期
  },
  fail: function() {
    //登录状态过期
    wx.login()
  }
})
```

这里还要注意，wx.login 得到的 code 只能使用一次，一旦你使用 code 换取了 openid 和 session_key，这个 code 就会马上失效，不能再次使用。当然，如果 5 分钟内这个 code 还没有被使用，那么也会失效。

虽然我们拿到了 openid 和 session_key，但并没有使用。拿到 openid 和 session_key 是用户登录要做的事情，其作用会在后续章节中逐步介绍。

12.3 用户信息校验

在 12.2 节中，我们学习了如何调用 wx.login 获取当前用户的 openid 以及本次登录的 session_key，本小节将介绍 session_key 的使用。

在 11.2 节中,我们调用 `wx.getUserInfo` 接口拿到了用户的明文基本信息数据和用户加密数据,并使用了明文数据。下面回顾一下 `wx.getUserInfo` 返回的数据:

- `userInfo` 用户信息对象。
- `rawData` 不包括敏感信息的原始数据字符串。
- `signature` 使用 `sha1(rawData + sessionkey)` 得到字符串,用于校验用户信息。
- `encryptedData` 包括敏感数据在内的完整用户信息的加密数据。
- `iv` 加密算法的初始向量。

在之前的 11.2 节中,我们使用了 `userInfo` 对象,包括 `userInfo` 和 `rawData` 在内的明文数据都可能存在被篡改的风险。如何知道明文数据是否被篡改了呢?

这个时候 `rawData` 和 `signature` 就可以发挥作用了。`rawData` 和 `signature` 用于校验用户数据到底有没有被篡改过(没有绝对安全的网络,数据极有可能被抓包或者通过其他方式篡改)。通常来说,想要实现这个校验必须在服务器编码才能进行。这需要小程序将获取的 `rawData` 和 `signature` 一并提交到服务器,由服务器完成校验工作。

校验的基本原理是:`rawData` 是用户原始明文数据, `signature` 是使用 `sha1(rawData + sessionkey)` 得到的字符串。理论上讲,如果数据没有被篡改,那么 `signature` 等于 `sha1(rawData + sessionkey)`;如果 `rawData` 或者 `signature` 被修改了,那么 `signature` 必然不再等于 `sha1(rawData + sessionkey)`。

是否存在 `signature` 和 `rawData` 同时被修改的情况呢?理论上是不可能的,因为 `session_key` 并不在网络上传输,篡改者不知道这个变量,被篡改且校验通过的概率很小。

有可能从 `signature` 中推算出 `session_key` 吗?理论上讲,这是不可能的。因为 `sha1` 算法是不可逆的,无法在已知 `rawData` 和 `signature` 的情况下推算出 `session_key`。不知道 `session_key` 就无法通过同时修改 `rawData` 和 `signature` 达到“欺骗校验的目的”。如果知道了 `session_key`,只需要修改 `rawData` 并重新用 `session_key` 计算一下新的 `sha1(rawData+session_key)` 就又可以以新的 `rawData` 等于新的 `sha1(rawData+session_key)` 了。这样,开发者就无法知道 `rawData` 是被修改过的。

这也是为什么官方文档一再强调,不要在网络上传输 `session_key`,而应该将其保存在服务器上使用,以降低 `session_key` 被泄露的风险。

`session_key` 有点类似于我们在数据库中保存用户密码时所使用的“盐(salt)”。在数据库保存用户密码时,并不是直接将用户的密码以明文的方式存放在数据库表中,通常都会使用 SHA-1 或者 MD5 算法将用户密码和 salt 随机字符串拼接在一起,重新计算一下再存入数据库中。被重新使用 SHA-1 或 MD5 算法计算的用户密码谁都不知道是什么,开发者也只能比对每次登录时输入的密码和数据库保存的密码是否一致,判断是否为合法用户,却无法知道密码到底是什么。

用户数据校验的流程图如图 12-5 所示。



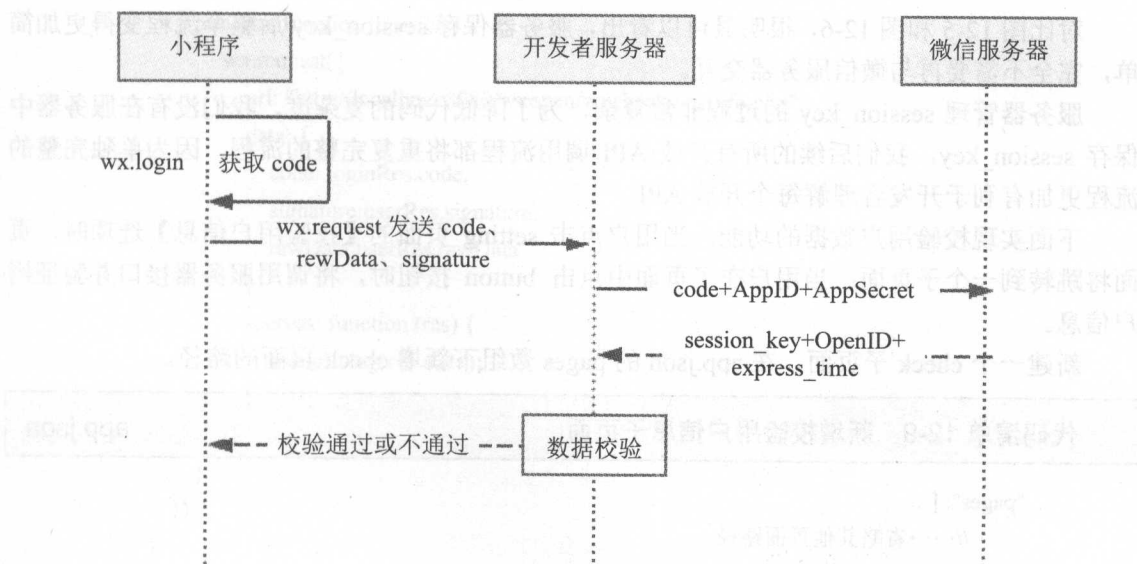


图 12-5 用户数据校验流程图

需要说明的是，在 12.2 节中，我们明确说明服务器是没有保存 session_key 的。因为我们需要拿到 session_key 才能进行用户数据校验，所以在上述流程图再一次重复了用户的登录流程。在真实的流程中，用户登录在 session_key 的有效时间内只应该执行一次，session_key 也应当被保存在服务器中。其实小程序只需要使用 wx.request 将 rawData 和 signature 发送到服务器即可，服务器无须使用 code 换取 session_key，直接做 SHA-1 签名比对即可。如图 12-6 所示为服务器已保存 session_key 的用户数据校验流程。

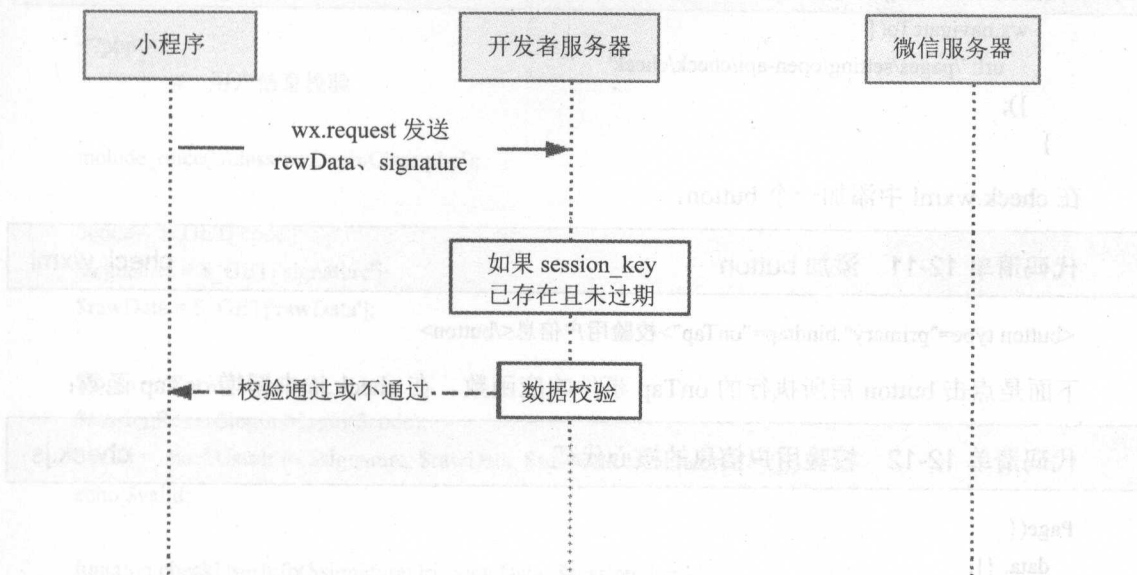


图 12-6 如果服务器已保存了 session_key 的用户数据校验流程

对比图 12-5 和图 12-6, 很明显可以看出, 服务器保存 session_key 后整个流程变得更加简单, 完全不需要再与微信服务器交互。

服务器管理 session_key 的过程非常复杂, 为了降低代码的复杂度, 我们没有在服务器中保存 session_key。我们后续的所有开放 API 调用流程都将重复完整的流程, 因为单独完整的流程更加有利于开发者理解每个开放 API。

下面实现校验用户数据的功能。当用户点击 setting 页面的【校验用户信息】选项时, 页面将跳转到一个子页面; 当用户在子页面中点击 button 按钮时, 将调用服务器接口并验证用户信息。

新建一个 check 子页面。在 app.json 的 pages 数组下新增 check 页面的路径。

代码清单 12-9 新增校验用户信息子页面

app.json

```
"pages": [
  //……省略其他页面路径

  "pages/setting/open-api/check/check"
]
```

接着, 在 setting.js 文件中编写点击子项的事件响应函数, 将页面导航至 check 子页面。

代码清单 12-10 跳转到 check 子页面

setting.js

```
check: function () {
  wx.navigateTo({
    url: '/pages/setting/open-api/check/check'
  });
}
```

在 check.wxml 中添加一个 button:

代码清单 12-11 添加 button

check.wxml

```
<button type="primary" bindtap="onTap">校验用户信息</button>
```

下面是点击 button 后所执行的 onTap 事件响应函数。在 check.js 中新增 onTap 函数:

代码清单 12-12 校验用户信息的核心代码

check.js

```
Page({
  data: {},
  onTap: function () {
    wx.login({
      success: function (loginRes) {
        wx.getUserInfo({
```




```

else{
    return "校验不通过";
}
}

```

在以上代码中，首先使用 code 调用微信服务器换取 session_key，随后使用 session_key 和 signature 校验用户发送过来的 rawData，最后返回校验结果。其中，login 已经在 12.2 节用户登录时编写完成，这里只需要复用这个类即可。图 12-7 所示是最终返回到小程序中的校验结果。

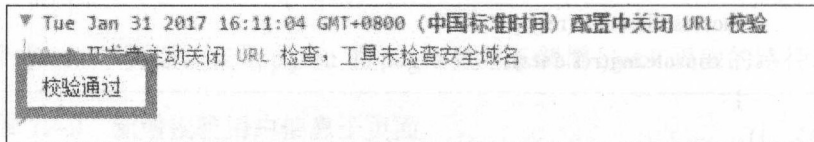


图 12-7 服务器返回的校验结果

我们可以简单测试校验不通过的情况，请在代码清单 12-12 中将 rawData 稍微修改一下，比如在调用 wx.request 时在 rawData 后面拼接一个字符串。

代码清单 12-14 修改 rawData 字符串

check.js

```

wx.request({
  url: "http://localhost:8080/wxopen/wxcheckuserinfo.php",
  data: {
    code: loginRes.code,
    signature: userRes.signature,
    rawData: userRes.rawData+"LOL"
  }
})

```

在发送到服务器的 rawData 字符串后面附加了一个字符串 LOL。再次运行代码将呈现如图 12-8 所示的输出结果。

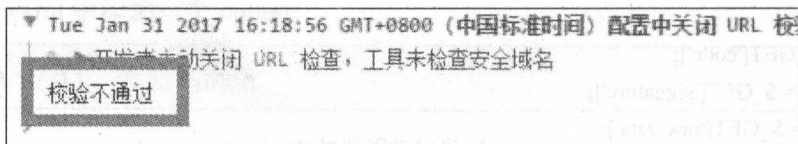


图 12-8 修改 rawData 后数据校验不通过

建议开发者在客户端使用用户明文数据时使用 rawData，而不要使用 userInfo。因为数据验证的是 rawData 有没有被篡改，而不是验证 userInfo 是否被篡改。至于微信能否确保 userInfo 和 rawData 的一致性，这个不得而知。建议开发者使用 rawData 作为用户的基本信息。



12.4 解析用户加密数据获取 openId 及 UnionId

在 12.3 节中我们完成了用户明文数据的校验工作,本节将学习解密用户的非明文用户信息。

调用 `wx.getUserInfo` 后将返回 `encryptedData` 和 `iv` 两个数据。`encryptedData` 是包括敏感数据在内的完整用户信息的加密数据, `iv` 用于解密这个数据。整个解密用户数据的过程同 12.3 节中用户信息校验的流程基本相同。不同的是,我们提交到服务器的数据是 `encryptedData` 和 `iv`, 而不是 `signature` 和 `rawData`。

首先,在 `app.json` 中注册 `decrypted` 子页面。

代码清单 12-15 解密用户加密信息页面

app.json

```
"pages": [
  //.....省略若干其他页面
  "pages/setting/open-api/decrypted/decrypted"
]
```

在 `setting.js` 页面中添加 `decrypted` 函数,用于跳转到 `decrypted` 子页面。

代码清单 12-16 跳转到 decrypted 页面的函数

setting.js

```
decrypted:function(){
  wx.navigateTo({
    url: '/pages/setting/open-api/decrypted/decrypted'
  });
}
```

在 `decrypted.wxml` 中添加一个 `button`。

代码清单 12-17 button

decrypted.wxml

```
<button type="primary" bindtap="onTap">获取用户加密数据</button>
```

下面编写用户数据解密的核心代码,在 `decrypted.js` 文件中编写 `onTap` 事件响应函数。

代码清单 12-18 向服务器发送数据解密请求

decrypted.js

```
Page({
  data: {},
  onTap: function () {
    wx.login({
      success: function (loginRes) {
        wx.getUserInfo({
```



```

    success: function (userRes) {
      wx.request({
        url: "http://localhost:8080/wxopen/wxdecryptuserinfo.php",
        data: {
          code: loginRes.code,
          encryptedData: userRes.encryptedData,
          iv: userRes.iv
        },
        success: function (res) {
          console.log(res.data);
        }
      })
    }
  })
}
})

```

以上代码将 `code`（用于获取 `session_key`）、`encryptedData` 和 `iv` 3 个参数发送到了 `wxdecryptuserinfo.php` 中。

服务器如何解密 `encryptedData` 数据呢？解密时需要 `session_key`、`iv`、小程序的 `AppId` 3 个变量参与。`session_key` 由 `code` 可以换取到，`iv` 由小程序客户端提交，`AppId` 本身就是一个固定的已知变量。

具体的解密算法较为复杂，但微信官方提供了包括 C++、NodeJS、PHP 和 Python 4 种语言的解密 SDK，我们只需要使用官方提供的 SDK 即可，无须自己编写解密算法。目前，官方没有提供 JAVA 和 C# 版本的 SDK，开发者可自行翻译。4 种语言解密 SDK 的下载地址为 <https://mp.weixin.qq.com/debug/wxadoc/dev/demo/aes-sample.zip>。

每种语言的接口名称都是一致的，请选择适合自己的语言。服务器的解密工作主要使用官方提供的 SDK 进行解密。

我们选择 PHP 版本的 SDK。PHP 版本的 SDK 主要包含以下 4 个文件：

- `wxBizDataCrypt.php`
- `pkcs7Encoder.php`
- `errorCode.php`
- `demo.php`

`wxBizDataCrypt.php` 是 SDK 的接口类，在内部调用了 `pkcs7Encoder.php`；`pkcs7Encoder.php` 提供基于 PKCS7 算法的加解密接口；`errorCode.php` 是错误码说明；`demo.php` 是使用解密 SDK 的示例，我们无须关心解密的细节（当然，如果有兴趣，开发者也可以看看源码研究学习一下）。



下面我们编写 wxDecryptUserInfo.php，该页面向小程序提供接口实现用户数据的解密，wxDecryptUserInfo.php 内部主要调用微信提供的 SDK 进行数据解密。首先，将微信提供的 3 个用于解密的 PHP 文件放在如图 12-9 所示的位置。

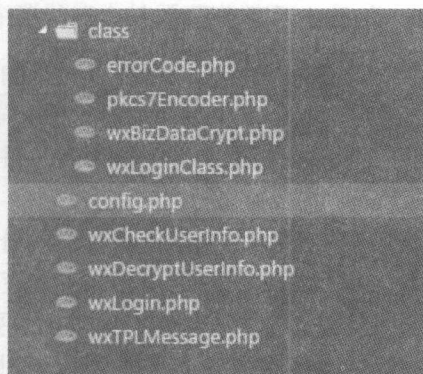


图 12-9 将微信解密 SDK 放置在 class 目录下

在 wxdecryptuserinfo.php 中编写以下代码：

代码清单 12-19 数据解密

wxdecryptuserinfo.php

```
<?php
# 解密用户非明文数据

include_once('./class/wxBizDataCrypt.php');
include_once('./class/WXLoginClass.php');
$config = include_once('config.php');

$code = $_GET['code'];
$encryptedData = $_GET['encryptedData'];
$iv = $_GET['iv'];
$wxAppID = $config['WXAPPID'];

$login = new WXLogin();
$sessionRes = $login->login($code);

$pc = new WXBizDataCrypt($wxAppID, $sessionRes['session_key']);
$errCode = $pc->decryptData($encryptedData, $iv, $data );

if ($errCode == 0) {
    echo $data;
} else {
    echo $errCode;
}
```

数据解密的代码同样首先使用 `code` 换取 `session_key`，接着实例化微信提供的 `WXBizDataCrypt`，实例化 `WXBizDataCrypt` 需要 `appid`、`session_key` 这两个参数，然后调用 `WXBizDataCrypt` 实例的 `decryptData` 方法实现数据的解密。最后判断 `errCode` 的取值，如果 `errCode` 为 0，就认为解密成功，将 `data` 返回小程序中。

以下是小程序最终得到的返回结果的数据示例结构：

代码清单 12-20 用户数据解密后的数据结构

```
{
  "openId": "OPENID",
  "nickName": "NICKNAME",
  "gender": GENDER,
  "city": "CITY",
  "province": "PROVINCE",
  "country": "COUNTRY",
  "avatarUrl": "AVATARURL",
  "UnionId": "UNIONID",
  "watermark": {
    "appid": "APPID",
    "timestamp": "TIMESTAMP"
  }
}
```

微信在数据的末尾添加了一个 `watermark` 水印字段，此字段可以用来校验应用数据的有效性。

- `appId` 敏感数据归属 `appId`，开发者可校验此参数与自身 `appId` 是否一致。
- `timestamp` 敏感数据获取的时间戳，开发者可以用于数据时效性校验。

12.5 模板消息

不同于前几节的示例代码，本节内容的示例代码需要以下 4 个条件才能成功运行：

- (1) 在真机上运行小程序（开发工具中拿不到 `formId`）。
- (2) 开发者服务器位于外网。
- (3) 外网地址已经加入小程序账号的可信域名列表中。
- (4) 服务器接口需要提供 `https` 支持（因为在真机上只能调用 `https` 接口）。

笔者一直认为，小程序应该充分借助微信的优势提升开发者的开发效率、降低开发成本并提升用户体验，比如可以借助微信消息体系向用户推送各类通知。信息触达率高是微信的一项优势，借助微信的信息触达渠道远比自建信息渠道要好得多。



微信订阅号可以一天群发一条消息，微信服务号一个月可以群发 4 条消息。和微信订阅号、服务号不同，小程序不可以主动向用户群发消息。这几乎切断了小程序的媒体属性，不利于开发者推广小程序。

笔者分析限制微信小程序群发能力的原因有以下 4 点：

(1) 在微信上的产品一向是极为克制的，从不盲目添加功能，凡是对用户体验有影响的功能一般都不会轻易开放。毕竟群发消息会对用户造成一定打扰。

(2) 在本书开始部分分析过，小程序主打的是用完即走的服务，在微信看来不应该具有媒体属性。

(3) 一定程度上是为了保护服务号，不让小程序对现有的产品体系造成太大冲击。

(4) 小程序现在处于萌芽状态，不能在微信上贸然开放太多功能。要知道泼出去的水是收不回来的，即使收回来也要付出较大代价。从策略上来看，先不给你，以后看情况再给你远比先给了你，再收回要高明很多。毕竟小程序的生态没有太多案例可以借鉴，一切都要“摸着石头过河”。

虽然小程序不可以主动向用户群发消息，但是微信提供了一个模板消息。需要注意的是，模板消息具有以下 4 个特点：

(1) 用户接收的模板消息位于微信消息的服务通知中，多个模板消息会被折叠进服务通知里。

(2) 模板消息是被动的，只有用户本人在小程序中有一定交互行为后，服务器才能够向用户推送模板消息。

(3) 模板消息有两种跳转功能，一种是可以进入发送模板的小程序里，这是模板消息的默认跳转，前提条件是必须是线上已发布的小程序；另一种是可以指定一个模板消息的跳转页面，跳转到小程序的内部，这种跳转在体验版或开发版中都可以成功跳转。

(4) 要成功发送模板消息，必须要有一个 `formId` 的 `id` 号，但经过测试发现开发工具中无法获取 `formId`，也就是说在开发工具中不能产生模板消息。只有在真机中才能拿到 `formId`，而 `formId` 是发送模板消息的关键。这也是为什么章节开头时提出一定要在真机中运行本示例。

上面我们提到了模板消息是被动的，只有用户本人在小程序中有一定交互行为后，服务器才能够向用户推送模板消息。下面来解释一下用户的什么行为才能让服务器可以向小程序推送一条模板消息。

(1) 用户在小程序内完成过支付行为，7 天内可允许开发者向用户推送有限条数的模板消息（一次支付可下发一条，多次支付可发条数独立，互相不影响）。

(2) 当用户在小程序内发生过提交表单行为且该表单声明为要发模板消息时，7 天内可允许开发者向用户推送有限条数的模板消息（一次提交表单可发一条，多次提交可发条数独立，相互不影响）。

在本示例中，以用户提交表单的行为编写模板消息示例代码。在编写代码前，首先要对模板消息发送的全部流程有一个清晰的认识。图 12-10 所示为用户提交表单类模板消息的完整流程。要注意的是，微信支付的模板消息与用户提交表单的模板消息流程不太一样，请不要混淆。

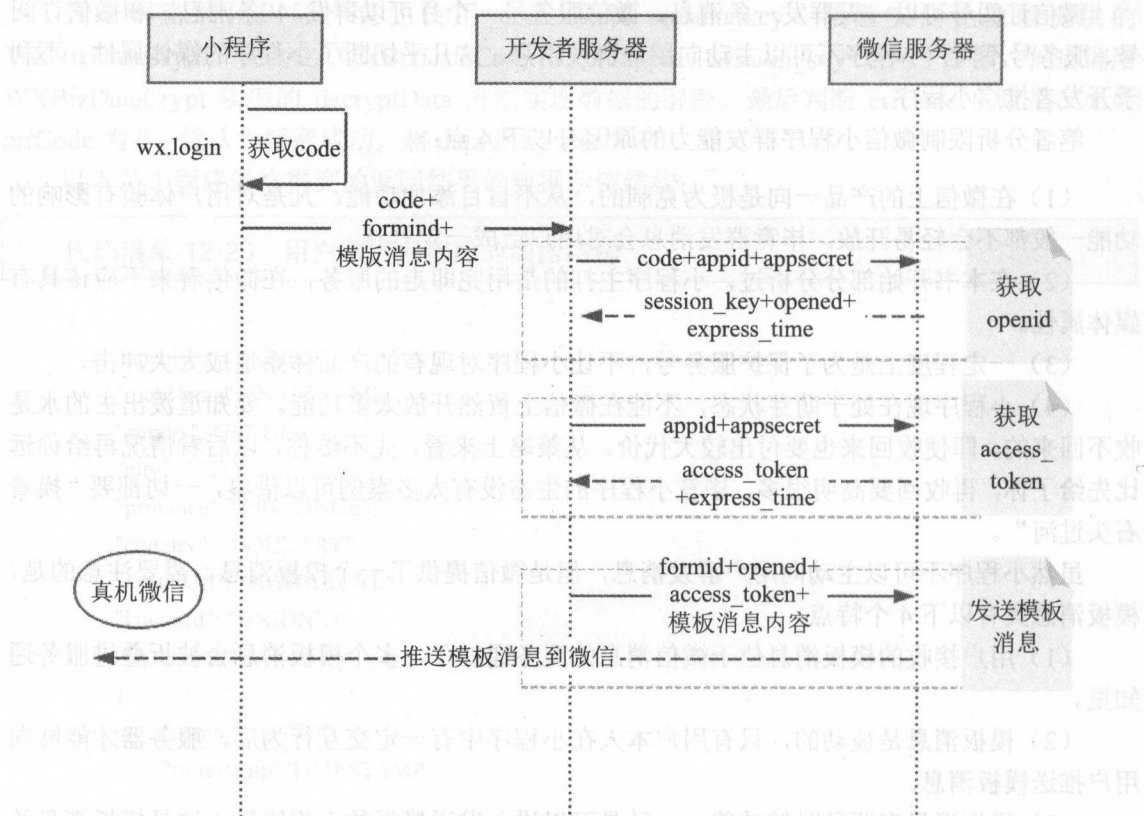


图 12-10 发送模板消息的完整流程

以上流程是建立在已经在小程序中拿到了 `formId` 且 `formId` 没有被使用的前提下。请开发者先大致浏览以上流程图，在后面具体编码时再回过头来对比这张流程图。发送模板消息需要知道用户的 `openid` (不知道 `openid` 就不知道到底发送给哪个用户)。这里要强调下，获取 `openid` 的流程不是必须的。如果用户已经在小程序中登录，服务器也保存了 `openid` 和 `session_key`，就没有必要再使用 `code` 换取 `openid`。

除此之外，发送模板消息的流程还需要一个 `access_token`。`access_token` 在微信服务号或者订阅号里经常被作为令牌使用，这里也被移植到了小程序中。`access_token` 也是有失效期的，开发者应该在真实的项目中像管理 `session_key` 一样管理 `access_token`。获取 `access_token` 同样需要携带微信小程序的 `appid` 和 `appsecret` 调用微信服务器。

流程图中没有标示出 `formId` 的获取过程，那么 `formId` 是怎么来的呢？

当用户提交表单时，表单提交函数的 `event` 事件对象中将包含一个 `formId`，这个 `formId` 只能使用一次，有效期为 7 天，一旦使用 `formId` 推送了一条模板消息，这个 `formId` 就不可以再次使用。如果还想推送模板消息，就只能等用户再一次提交表单并产生新的 `formId`。

最后还有一个问题，模板的格式怎么定义？先来看示例代码最终推送的模板消息示意图，如图 12-11 所示。

小程序提供了许多种类型的模板，每个模板都有一个模板编号。可以在小程序公众账号“模板消息”的“模板库”中选择模板和获取模板编号，如图 12-12 所示。



图 12-11 最终推送的模板消息示意图

ID	标题	预览关键词	使用人数	操作
AT0002	购买成功通知	购买地点, 购买时间, 物品名称, 交易单号, 购...	4133	选用
AT0007	订单发货提醒	快递公司, 发货时间, 购买时间, 物品名称, 发...	966	选用
AT0009	订单支付成功通知	单号, 金额, 下单时间, 物品名称, 订单号码, ...	869	选用
AT0004	交易提醒	交易时间, 交易类型, 交易金额, 商户详情, 商...	589	选用
AT0005	付款成功通知	物品名称, 付款时间, 付款金额, 地点, 产品名...	414	选用
AT0008	待付款提醒	单号, 金额, 下单时间, 物品名称, 商品详情, ...	340	选用

图 12-12 小程序提供了大量模板

示例代码中选择的是【购买成功通知】模板，这个模板的详情如图 12-13 所示。

	模板ID	LKG10a	复制
	标题	购买成功通知	
购买成功通知 2017年02月	关键词	购买地点 {{keyword1.DATA}} 购买时间 {{keyword2.DATA}} 物品名称 {{keyword3.DATA}} 交易单号 {{keyword4.DATA}}	
购买地点	TIT工厂		
购买时间	2016年6月6日		
物品名称	咖啡		
交易单号	23432132		

图 12-13 购买成功通知模板详情

我们需要关注模板中的两个关键点：模板 ID 和关键词。模板 ID 表示需要向开发者发送哪种类型的模板消息。关键词是这个模板消息可以发送的内容，比如购买成功通知模板包含购买地点、购买时间、物品名称和交易单号 4 个关键词。在后面的示例代码中我们将看到模板 ID 和关键词的使用方法。

如果在小程序提供的模板库中没有你想要的模板，那么可以向微信团队提出新模板申请。申请新模板有一些约束，请参考以下文档 <https://mp.weixin.qq.com/debug/wxadoc/dev/api/notice.html#接口说明>。该文档中有一个“审核说明”，详细描述了申请新模板的一些要求。

了解以上内容后，开始编写 Orange Can 项目中的模板消息示例代码。首先，在 app.json 的 pages 数组中注册 tpl-message 模板消息子页面。

代码清单 12-21 新建模板消息子页面

app.json

```
"pages": [
  // .....省略若干其他页面
  "pages/setting/open-api/tpl-message/tpl-message"
]
```

在 setting.js 页面中编写跳转到 tpl-message 模板消息子页面的函数 tplMessage。

代码清单 12-22 跳转到 tpl-message 页面

setting.js

```
tplMessage:function(){
  wx.navigateTo({
    url: '/pages/setting/open-api/tpl-message/tpl-message'
  });
}
```

在 tpl-message.wxml 中编写模板消息页面的骨架代码。

代码清单 12-23 模板消息页面的骨架代码

tpl-message.wxml

```
<form bindsubmit="formSubmit" report-submit="true">
  <input name="place" type="text" placeholder="购买地点: " />
  <picker name="date" mode="date" value="{{date}}"
    start="2017-1-1" end="2020-12-31"
    bindchange="bindDateChange">
    购买时间: {{date}}
  </picker>
  <input name="name" type="text" placeholder="物品名称:" />
  <input name="id" type="text" placeholder="交易单号: " />
  <button formType="submit">Submit</button>
</form>
```



代码清单 12-24 模板消息页面的样式

tpl-message.wxss

```

input {
  height: 100px;
  width: 100%;
  background-color: #f6f6f6;
  margin-bottom: 30rpx;
}

button {
  background-color: #4a6141;
}

picker {
  width: 100%;
  background-color: #f6f6f6;
  margin-bottom: 30rpx;
  padding: 19rpx 26rpx;
}

```

在继续编写模板消息核心业务逻辑代码前,先插入一节介绍模板消息页面中的 form 表单。

12.6 form 表单及 picker 组件

在 12.5 节的模板消息页面中,使用<form>标签构建了一个表单。form 表单有什么作用呢?

form 表单中有 3 个 input 和 1 个 picker 组件,假如想获取这 4 个组件的用户输入值,怎么办呢?一种笨办法是在每个组件的 bindchange、bindconfirm 等事件中获取该组件的用户输入值,比如我们之前多次使用的 input 组件就是通过此类组件的特殊事件获得的。

假如页面的表单元素非常多,这样一个一个获取太过于烦琐。form 标签的作用就是让开发者可以一次性获取所有表单组件输入值。form 表单有以下 3 个重要属性:

- report-submit Boolean 类型。是否返回 formId 用于发送模板消息。
- bindsubmit EventHandle 类型。指定一个响应函数,当 form 表达内部类型为 submit 的 button 被点击时,将触发这个响应函数。
- bindreset EventHandle 类型。指定一个响应函数,当 form 表达内部类型为 reset 的 button 被点击时,将触发这个响应函数。

当用户点击 form 表达内部类型为 submit 的 button 后,将执行 bindsubmit 属性所指定的响应函数。在响应函数的 event 事件对象中将可以获取 form 下所有表单元素的用户输入值。

注意,form 表单只会提交 6 种类型组件的用户输入值:

- <switch/>
- <input/>
- <checkbox/>
- <slider/>
- <radio/>
- <picker/>

同时, report-submit 属性也是一个非常重要的属性。如果此属性为 true, 那么触发 bindsuit 后, event 事件对象中将包含一个 formId, 这就是要用于发送模板消息所需要的 formId。要注意的是, 经过笔者测试, 在开发工具中 formId 是一个 mock 字符串: the formId is mock one。这并非是一个可以使用的 formId。只有在真机上才能获取真实的 formId。

下面介绍 picker 组件。小程序的表单类元素有很多, 但其用法和属性都比较简单, 官方文档中的说明也比较清楚。建议开发者在用到某个表单组件时具体查阅。

Picker 组件是从底部弹起的滚动选择器, 现在支持 3 种选择器, 通过 mode 属性来区分, 分别是普通选择器、时间选择器和日期选择器, 默认是普通选择器。下面来看 picker 组件的真机效果图, 如图 12-14 所示。

mode=selector 时, picker 为普通选择器。这是一种自定义类型的选择器, 如果需要自定义选择内容, 请使用这种类型的选择器。这种类型的选择器有以下 5 种属性:

- range 数组类型。mode 为 selector 时, range 属性才有效。
- range-key String 类型。当 range 是一个 Object Array 时, 通过 range-key 指定 Object 中 key 的值作为选择器的显示内容。
- value Number 类型, 默认值为 0。value 的值表示选择了 range 中的第几个 (下标从 0 开始)。
- bindchange EventHandle 类型。value 改变时触发 change 事件。
- disabled Boolean 类型, 默认值为 false。表示是否禁用。

mode 为 time 时, picker 为时间选择器。这种模式的 picker 用于选择时间, 具有以下 5 个属性:

- value String 类型。表示选中的时间, 格式为 "hh:mm"。
- start String 类型。表示有效时间范围的开始, 字符串格式为 "hh:mm"。
- end String 类型。表示有效时间范围的结束, 字符串格式为 "hh:mm"。
- bindchange EventHandle 类型。value 改变时触发 change 事件。
- disabled Boolean 类型, 默认值为 false。表示是否禁用。

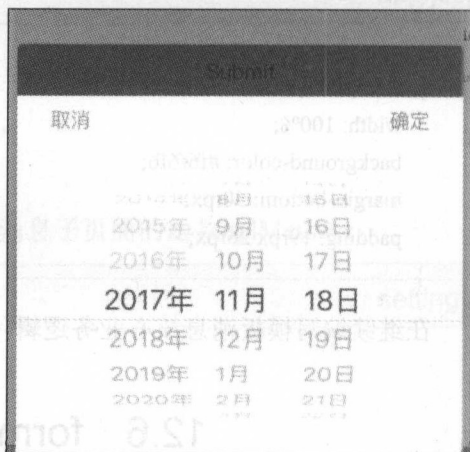


图 12-14 picker 组件的真机效果图

当mode为date时，这种模式的picker用于选择日期，具有以下6个属性：

- value String 类型，默认值是 0。表示选中的日期，格式为"YYYY-MM-DD"。
- start String 类型。表示有效日期范围的开始，字符串格式为"YYYY-MM-DD"。
- end String 类型。表示有效日期范围的结束，字符串格式为"YYYY-MM-DD"。
- fields String 类型。默认值是 day，有效值为 year、month、day，表示选择器的粒度。
- bindchange EventHandle 类型。当 value 改变时触发 change 事件。
- disabled Boolean 类型。默认值是 false。表示是否禁用。

以上有 3 种类型的属性文档都讲述的非常明白，我们只对几个存在误区理解的属性做一些解释。

首先是 value 属性，picker 的 value 和 input 的 value 不同。picker 的 value 不是显示在组件上的文本，而是定义 picker 默认的选择项目，比如将 value 设置为"2017-11-18"，那么打开 picker 时默认将停留在 2017-11-18 这个选项上。

其次是 start 和 end 属性，start 和 end 属性只有当 mode 为 time 和 mode 为 date 时才有。这两个属性并不是指定 picker 组件的最大选择范围，而是指定可选范围。举个例子，将 start 设置为 2017-1-1，end 设置为 2020-12-31，还是可以看到超出这个时间范围的年月日，但却无法选择这些超出范围的日期。

12.7 发送模板消息

下面继续编写模板消息功能。在之前的章节中，我们编写了 tpl-message 页面的骨架和样式，现在这个页面如图 12-15 所示（购买时间选项将在编写 tpl-message.js 文件后出现）。

在模板消息页面，我们需要采集一些用户的输入值，这些输入值将被提交到服务器中用于发送模板消息，购买地点、购买时间、物品名称和交易订单号刚好对应【购买成功通知】消息模板的 4 个关键词。

接着编写 formSubmit 函数，点击页面的 Submit 按钮后将执行此函数。注意，这个事件响应函数名

图 12-15 tpl-message 模板消息页面的效果图

在 tpl-message.js 文件中编写以下代码：

代码清单 12-25 编写 tpl-message 的业务逻辑

tpl-message.js

```
Page({
  data: {
    date: "2017-11-18"
```

```

    },
    bindDateChange: function (event) {
      this.setData({
        date: event.detail.value
      })
    },
    formSubmit: function (event) {
      wx.login({
        success: function (loginRes) {
          wx.request({
            url: 'https://xxxxxxx.com/wxopen/wxtplmessage.php?code='
              + loginRes.code,
            data: {
              formId: event.detail.formId,
              formData: event.detail.value
            },
            method: 'POST',
            success: function (res) {
              console.log(res.data);
            }
          })
        }
      })
    }
  })
}
})

```

首先，调用 `wx.login` 方法得到了 `code`。接着，使用 `wx.request` 方法调用服务器接口。注意，代码中的 `wx.request` 的 `url` 设置的是一个示例 `url`。前面我们讲过本节的示例必须在真机上运行才能获取 `formId`，真机运行必须调用 `https` 接口。除此之外，接口所在的域名必须加入小程序账号的可信域名列表中。同时要注意，我们将 `code` 附加在了这个 `url` 的 `query` 参数中。

这里我们使用的是 `POST` 请求，而不是 `GET` 请求。`POST` 的数据在 `wx.request` 的 `data` 中。`data` 包含两个属性，即 `formId` 和 `formData`。`formData` 的取值来自于 `event.detail.value`，包含用户输入的购买地点、购买时间、物品名称和交易单号 4 个值。

以上是小程序中的全部代码，接着我们需要编写服务器的 `PHP` 代码。

首先，在 `config.php` 中增加两个配置项，一个是 `TOKENURL`，用于获取 `access_token` 的地址；另一个是 `TPLMSGURL`，是微信服务器发送模板消息的 `URL`。

代码清单 12-26 添加 TOKEN 和模板消息的访问地址

config.php

```

<?php
return array(
  'WXAPPID' => 'your app id',
  'WXAPPSECRET' => 'your app secret',

```



```

'LOGINURL' => "https://api.weixin.qq.com/sns/jscode2session?".
               "appid=%s&secret=%s&js_code=%s&grant_type=authorization_code",
'TOKENURL' => "https://api.weixin.qq.com/cgi-bin/token?".
               "grant_type=client_credential&appid=%s&secret=%s",
'TPLMSGURL' => "https://api.weixin.qq.com/cgi-bin/message/wxopen/template/send?".
               "access_token=%s"
);

```

获取 access_token 是一个 HTTP GET 请求，需要附带小程序的 appid 以及 appsecret；发送模板消息是一个 HTTP POST 请求，需要 access_token（access_token 将附加在接口 url 的 query 参数中）以及模板消息的一系列参数（位于 POST 的 body 中）。注意将 WXAPPID 和 WXAPPSECRET 换成自己小程序的 appid 和 appsecret。

接着新增一个 http.php 文件，位于根目录的 class 文件夹下。http.php 文件包含两个方法，即 curl_post 和 curl_get，用于发送 http 请求。

代码清单 12-27 curl_post 及 curl_get

http.php

```
<?php
```

```

function curl_post($url,array $params = array()){
    $data_string = json_encode($params);
    $ch = curl_init();
    curl_setopt($ch,CURLOPT_URL,$url);
    curl_setopt($ch, CURLOPT_HEADER, 0);
    curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
    curl_setopt ($ch, CURLOPT_CONNECTTIMEOUT, 10);
    curl_setopt($ch, CURLOPT_POST, 1);
    curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, false);
    curl_setopt($ch, CURLOPT_POSTFIELDS, $data_string);
    curl_setopt($ch, CURLOPT_HTTPHEADER,
        array(
            'Content-Type: application/json'
        )
    );
    $data = curl_exec($ch);
    curl_close($ch);
    return($data);
}

```

```

function curl_get($url){
    $ch = curl_init();
    curl_setopt ($ch, CURLOPT_URL, $url);
    curl_setopt ($ch, CURLOPT_RETURNTRANSFER, 1);

```

```

        curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, false);
        curl_setopt($ch, CURLOPT_CONNECTTIMEOUT, 10);
        $file_contents = curl_exec($ch);
        curl_close($ch);
        return $file_contents;
    }

```

最后，编写 wxTPLMessage.php 文件的代码，wxTPLMessage.php 位于根目录下。

代码清单 12-28 模板消息的接口代码

wxTPLMessage.php

```

<?php
    # 模板消息示例代码

    include_once('./class/http.php');
    include_once('./class/wxLoginClass.php');
    $config = include_once('config.php');

    //接收参数
    $data = file_get_contents("php://input");
    $data = json_decode($data,true);
    $code = $_GET['code'];

    //获取 openId
    $login = new WXLogin();
    $loginRes = $login->login($code);
    $openId = $loginRes['openId'];

    //获取 access_token
    $tokenUrl = $config['TOKENURL'];
    $tokenUrl = sprintf($tokenUrl, $config["WXAPPID"], $config["WXAPPSECRET"]);
    $access_token = curl_get($tokenUrl);
    $access_token = json_decode($access_token,true);

    //发送模板消息
    $tplMSG = array(
        'touser' => $openId,
        'form_id' => $data['formId'],
        'page' => 'pages/setting/setting',
        //填写自己小程序账号的模板 ID
        'template_id' => 'your template_id',
        'data' => array(
            'keyword1'=>array(

```




```

        "value" => $data['formData']['place'],
        "color" => '#333'
    ),
    'keyword2'=>array(
        "value" => $data['formData']['date'],
        "color" => '#333'
    ),
    'keyword3'=>array(
        "value" => $data['formData']['name'],
        "color" => '#333'
    ),
    'keyword4'=>array(
        "value" => $data['formData']['id'],
        "color" => '#333'
    )
),
"emphasis_keyword" => "keyword3.DATA"
);

$tplMSGUrl = sprintf($config['TPLMSGURL'],$access_token['access_token']);
$msgResult = curl_post($tplMSGUrl, $tplMSG);
echo($msgResult);

```

wxTPLMessage.php 是处理小程序提交模板消息内容的核心代码。首先，代码从 URL 的 query 参数和 http body 中分别获得 code 和模板消息内容。请开发者在阅读代码时再回顾一下图 12-10 的流程图。

拿到 code 后还是先调用 login，不同于前几节，此次 login 是为了获取用户的 openId。

接着，使用小程序的 appId 和 secret 去微信服务器换取 access_token。

最后，将 openId、access_token 及客户端发送来的模板消息内容提交到微信服务器，如果数据正常，那么微信会向指定 openId 的微信用户发送一条消息。

我们重点看一下\$tplMSG 关联数组下的元素。

- **Touser** 指定模板接收者的 openId。
- **template_id** 模板消息的 id 号，这个编号可以直接从小程序公众号账号中获取。
- **page** page 指定了用户在收到模板消息后点击模板消息将跳转的小程序页面
- **data** 模板消息的内容，我们选择的模板拥有 4 个关键词，每个关键词中包含 value 和 color 两个属性，分别指定该关键词的内容以及关键词的显示颜色。
- **emphasis_keyword** 只能指定一个关键词，这个关键词将被“强调”。被强调的关键词将在模板消息中居中、放大、加重。

真机上接收的模板消息效果图如图 12-16 所示。

模板消息是实现产品完整闭环逻辑的重要环节，请开发者对这个功能给予足够重视。

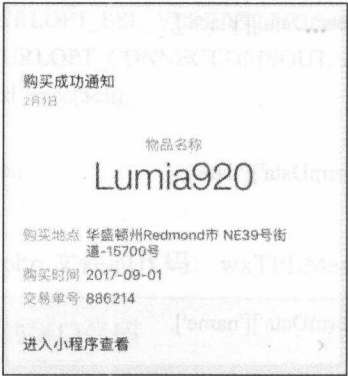


图 12-16 模板消息真机效果图

12.8 微信支付

微信支付是开放接口中最为重要的接口。小程序的微信支付基本同微信 JSAPI 中的微信支付相同。如果你有 JSAPI 中开发微信支付的经验，小程序的支付相当简单。

本小节我们来实现小程序中的微信支付。首先来看微信小程序的支付流程图(见图 12-17)，该图来自于官方 API 文档。

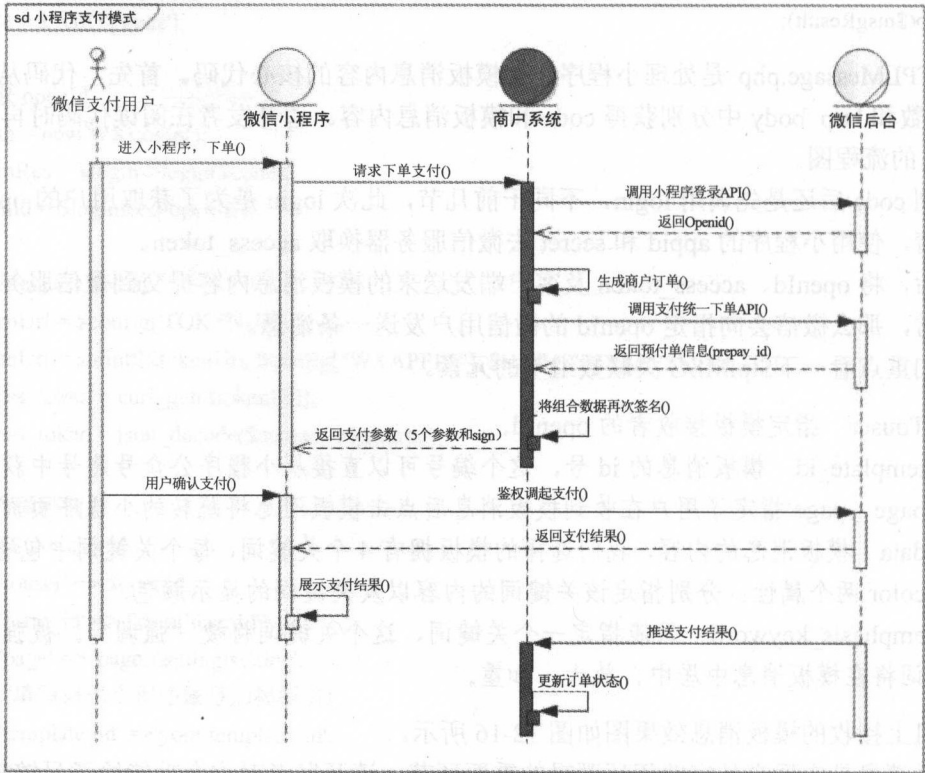


图 12-17 微信小程序支付流程图

文档地址为 https://pay.weixin.qq.com/wiki/doc/api/wxa/wxa_api.php?chapter=7_4&index=2。

支付流程看似比较复杂,但微信提供了一个 SDK,使用 SDK 基本上不需要编写太多支付相关的代码。事实上,在真实项目中,支付相关的代码复杂的不是支付本身,而是我们自己的业务逻辑。

下面来看完成一次支付的流程。这个流程是微信支付的最简流程。流程中不包括开发者自己的业务逻辑。

步骤 01 首先,小程序客户端调用开发者自己的服务器,将一系列订单信息发送到服务器,比如商品的 id 等信息。

步骤 02 开发者服务器接收被购买商品的信息后,调用微信服务器的统一下单 API,生成一个预付单,并将预付单信息返回开发者服务器。统一下单 API 需要用户的 openId。如果你已经在服务器中保存了 openId (之前几节我们已经反复给出了获取 openId 的示例代码),就不再需要获取用户的 openId 了;如果你没有当前支付用户的 openId,那么需要在第一步中携带 code,以方便在这一步中使用 code 换取用户的 openId。本小节示例中,我们假设服务器没有用户的 openId。

步骤 03 开发者服务器需要对预付单的信息签名,并将预付单信息和签名一起返回小程序客户端。

步骤 04 小程序客户端在收到预付单信息及签名后,再调用 `wx.requestPayment`,将预付单信息和签名一起提交到微信服务器。

步骤 05 微信服务器会验证这些预付单信息,如果验证通过,那么小程序将拉起支付界面(在开发工具中,这一步首先会弹出一个二维码,开发者扫描这个二维码将在开发者的微信中拉起支付界面)。

步骤 06 支付完成后,微信会主动调用开发者服务器将支付结果推送到开发者服务器中,开发者可根据支付结果处理自己的业务逻辑。这一步需要开发者有自己的外网服务器,否则微信无法推送通知。

以上 6 个步骤中,第二步是关键。

下面我们实现这 6 个步骤,在 Orange Can 项目中完成一次小程序的支付。首先,在 Orange Can 项目中新增一个页面 wx-pay。在 app.json 的 pages 数组下新增该页面的路径。

代码清单 12-29 新建 wx-pay 页面

app.json

```
"pages": [
  "pages/setting/open-api/wx-pay/wx-pay"
]
```

接着,在 setting.js 中编写跳转到 wx-pay 支付页面的跳转函数。

代码清单 12-30 从 setting 页面跳转到 wx-pay 页面

setting.js

```
wxPay: function () {
  wx.navigateTo({
```

```
url: '/pages/setting/open-api/wx-pay/wx-pay'
```

在 wx-pay.wxml 中增加一个按钮用来触发微信支付。

代码清单 12-31 新增支付按钮

wx-pay.wxml

```
<button type="primary" bindtap="onTap">微信支付</button>
```

以上准备工作做好后，就可以编写客户端支付的相关代码了。在 wx-pay.js 中新增 onTap 事件响应函数，用户点击【微信支付】按钮后，将触发 onTap 函数。

代码清单 12-32 微信支付关键代码

x-pay.js

```
Page({
  data: {},
  onTap: function () {
    wx.login({
      success: function (res) {
        wx.request({
          url: "http://127.0.0.1:8080/wxopen/wxpay.php",
          data: {
            code: res.code
          },
          success: function (res) {
            var preData = res.data;
            console.log(preData);
          }
        })
      }
    })
  }
})
```

回顾一下我们之前总结的 6 个步骤，在正式拉起支付界面前，需要向自己的服务器请求预支付订单信息。只有获取预支付订单信息，才能在小程序中拉起支付界面。由于预支付订单的生成必须知道用户的 openId，因此再一次调用了 wx.login 获取 code，将 code 发送到自己的服务器中。由于本项目不是真实项目，因此在这一步中没有提交任何商品信息，开发者在开发真实项目时可以将自己商品的信息同 code 一起发送到服务器中。

要特别注意的是，服务器的接口地址 url 一定不要使用 localhost，而应该使用 127.0.0.1。具体原因是服务器中的微信 SDK 在获取客户端 IP 时可能会将 localhost 获取成 "::1" 这样的字符串（可能是开启了 IPv6 支持引起的），这个字符串是微信 SDK 不允许的。当然，在真实的运行环境中不存在这个问题。建议开发者在调试时使用 127.0.0.1。

代码的 `wx.request` 的 `success` 中将接收服务器返回的预支付订单信息。获取预支付订单信息后，调用 `wx.requestPayment(OBJECT)` 即可拉起微信支付界面。在继续编写调用 `wx.requestPayment(OBJECT)` 前，我们先来编写服务器代码，没有服务器代码是无法返回预支付订单信息的。

小程序支付的服务器部分基本上不需要编写任何代码，微信已经提供了一整套 SDK 供我们调用。微信支付服务的 SDK 提供了 Java、C# 和 PHP 三个版本。SDK 包下载地址为 https://pay.weixin.qq.com/wiki/doc/api/jsapi.php?chapter=11_1。

下载并解压 sdk 包。我们需要使用 SDK 包根目录下 `lib` 文件夹里的 5 个文件：

- `WxPay.Api.php`
- `WxPay.Config.php`
- `WxPay.Data.php`
- `WxPay.Exception.php`
- `WxPay.Notify.php`

其中，`WxPay.API.php` 是 SDK 接口，`Wxpay.Config.php` 是配置文件，`WxPay.Data.php` 定义了一组微信支付中需要用到的基本对象，最后两个文件分别是错误异常类及接收回调通知的处理类。

在服务器代码的根目录下新增一个 `wxpay` 文件夹，将这 5 个文件拷贝到 `wxpay` 目录下。

开发者必须将自己的一些小程序账号信息配置到 `WxPay.Config.php` 中。

- `APPID` 小程序 `appid`。
- `MCHID` 商户号 `id`。
- `KEY` 商户的支付密钥。
- `APPSECRET` 小程序的 `appsecret`。

`MCHID` 和 `KEY` 在开发者商户账户的开通邮件中可以查看。`APPID` 和 `APPSECRET` 在小程序公众账号中可以获得。

配置完 `WxPay.Config.php` 后，新建 `wxPay.php` 文件作为接口文件供 `Orange Can` 项目调用。`wxPay.php` 主要作用是调用微信的统一下单接口，得到预支付订单信息，最后将数据加上签名再返回小程序中。由于只是示例代码，因此代码中没有添加容错处理，开发者请以理解微信支付下单流程为主。开发自己的项目时请合理处理异常并加强安全性。

下面编写 `wxPay.php` 中的代码。

代码清单 12-33 PHP 中关于支付的核心代码

`wxPay.php`

```
<?php
```

```
require_once 'wxpay/WXPay.Api.php';
require_once 'wxpay/WXPay.data.php';
require_once 'class/wxLoginClass.php';
```



```
$code = $_GET['code'];
```

```
//模拟订单信息
```

```
$payArray = array(
    'out_trade_no' => (string)time(),
    'trade_type' => 'JSAPI',
    'total_fee' => 1,
    'body' => '腾讯-红橙黄绿青蓝紫钻',
    'notify_url' => 'localhost:8080'
);
```

```
//使用 code 换取用户的 openId
```

```
$login = new WXLogin();
$loginRes = $login->login($code);
```

```
//构建预支付订单所需要的信息
```

```
$payData = new WxPayUnifiedOrder();
$payData->SetOut_trade_no($payArray['out_trade_no']);
$payData->SetTrade_type($payArray['trade_type']);
$payData->SetTotal_fee($payArray['total_fee']);
$payData->SetBody($payArray['body']);
$payData->SetOpenId($loginRes['openId']);
$payData->SetNotify_url($payArray['notify_url']);
```

```
//获取预支付订单信息
```

```
$orderNo = WxPayApi::unifiedOrder($payData);
$paySignData = prepareSignData($orderNo);
echo(json_encode($paySignData));
```

```
//将数据添加 MD5 签名
```

```
function prepareSignData($orderNo){
    $jsApiPayData = new WxPayJsApiPay();

    $config = include 'config.php';
    $jsApiPayData->SetAppid($config['WXAPPID']);
    $jsApiPayData->SetTimeStamp(time());

    $rand = md5(time().mt_rand(0,1000));
    $jsApiPayData->SetNonceStr($rand);

    $jsApiPayData->SetPackage('prepay_id='.$orderNo['prepay_id']);
```



```

$jsApiPayData->SetSignType('md5');
$sign = $jsApiPayData->MakeSign();

$rawValues = $jsApiPayData->GetValues();
$rawValues['paySign'] = $sign;
return $rawValues;
};

```

代码中的注释很清晰地指示了每段代码的作用。模拟订单信息这一段中，我们虚拟了一件商品的订单信息。这个订单信息包括以下 5 个参数：

- `out_trade_no` 商品订单编号。
- `trade_type` 在小程序中这是一个固定值，必须传入 JSAPI。
- `total_fee` 订单总金额，单位为分。
- `body` 商品的简单描述。
- `notify_url` 支付完成后，微信服务器将支付结果回调通知这个 url。

以上代码中，仅仅定义了几个必须的订单参数，微信支付的订单信息远远不止这 5 个参数。总共有十几个，其中过半参数都是必填选项，其余为选填选项，开发者应该根据自己的业务逻辑填写更详细的订单信息。本示例中定义的这几个订单数据仅保证支付能够执行成功，实际上还有更多商品描述和支付控制选项。具体请参考下面的官方文档地址：https://pay.weixin.qq.com/wiki/doc/api/wxa/wxa_api.php?chapter=9_1

订单信息的必填选项高达 10 个，为什么我们只指定这 5 个参数？因为小程序的 SDK 中会将一些其他必填选项自动加载，比如小程序的 `appid`、商户号等配置项已经在 `WxPay.Config.php` 中配置过了。

这里还要对 `notify_url` 做一个说明，其实这个选项也应该在 `WxPay.Config.php` 中配置。但经笔者测试，官方的 SDK 中 `WxPay.Config.php` 并没有这个配置项，所以我们选择在外定义 `notify_url`。如果开发者在这里既不定义 `notify_url`，又不在 `WxPay.Config.php` 中配置 `notify_url`，那么代码运行会报错。

为了方便测试，我们的服务器尽量部署在本地，所以无法接收微信的回调通知，`notify_url` 随便填写了一个地址。开发者在真实的项目开发中要将 `notify_url` 配置成能够从外网访问到的地址，以便接收微信支付结果的通知。

使用 `code` 换取用户 `openId` 的意图非常明显，就是从微信服务器拿到用户的 `openId`。

接着，构建预支付订单所需要的信息。`WxPayUnifiedOrder` 是微信支付 SDK 中定义的一个类，我们将使用准备好的订单信息数据及 `openId` 构建一个 `WxPayUnifiedOrder` 对象。

构建完 `WxPayUnifiedOrder` 对象后，调用 `WxPayApi` 的 `unifiedOrder` 方法并将 `WxPayUnifiedOrder` 对象作为参数传递进去。`unifiedOrder` 方法的内部将调用微信统一下单 API，并返回结果。返回结果包括 `appid`、`mch_id`（商户号）、`nonce_str`（随机字符串）、`sign`（签名）、`result_code`（业务结果）、`trade_type`（交易类型）、`prepay_id`（预支付交易会话表示）等。其中，`prepay_id` 尤其重要，小程序客户端就是依靠这个变量成功拉起微信支付。



也可以不使用微信 SDK，自行调用微信统一下单 API: <https://api.mch.weixin.qq.com/pay/unifiedorder>

拿到 `prepay_id` 预支付订单信息后不能直接将这些信息返回客户端。虽然这些信息已经完全包含了小程序成功调起支付界面的全部信息，但是这些信息还需要被签名。如果想将这些信息返回客户端，在客户端签名，是不可以的，因为签名需要商户号的密钥，将商户号密钥传输到客户端是非常不安全的行为。

首先，我们来看小程序拉起微信支付界面需要调用什么方法、传入哪些数据。只有知道客户端需要什么方法和参数，才能有选择的为数据签名。小程序在客户端中需要调用 `wx.requestPayment` 方法才能够拉起微信支付，以下是这个方法所需要的参数：

- `timestamp` String 类型，时间戳是从 1970 年 1 月 1 日 00:00:00 至今的秒数，即当前时间。
- `nonceStr` String 类型，随机字符串，长度为 32 个字符以下。
- `package` String 类型，统一下单接口返回的 `prepay_id` 参数值，提交格式如：`prepay_id=*`。
- `signType` String 类型，签名算法，暂时只支持 MD5。
- `paySign` String 类型，签名。

以上参数说明来自于官方说明文档。很多开发者看到这里都会下意识地认为从服务器将以上参数返回回来签名就可以了，事实上 `paySign` 签名是在服务器里完成后才返回的。如果签名在客户端完成，签名就没有什么意义了。我们无须关心签名的具体算法，在服务器的微信 SDK 中已经集成了签名计算的方法，开发者只需直接调用即可。当然，如果了解签名的具体算法，可以到：https://pay.weixin.qq.com/wiki/doc/api/jsapi.php?chapter=4_3 了解。

这里要特别强调，以上 5 个参数是小程序调用 `wx.requestPayment` 拉起微信支付的必要参数，但不是生成 `paySign` 签名的全部参数，这一点尤其重要。生成 `paySign` 签名需要以下 5 个参数：

- `appId` 小程序 `appId`。
- `timestamp` 时间戳。
- `nonceStr` 随机字符串。
- `package` 统一下单接口返回的 `prepay_id`。
- `signType` 签名算法，暂时只支持 MD5。

很明显，调用 `wx.requestPayment` 不需要 `appId`，但签名中需要加入 `appId`（如果被签名的数据和显示传递的数据一样，签名就没有意义了）。

我们来整理下思路。现在服务器已经拿到了 `prepay_id`，服务器需要生成一个 `timestamp` 时间戳、一个 `nonceStr` 随机字符串，再结合 `appId`、`prepay_id` 和 `signType`，使用这 5 个参数一起生成一个签名。随后将签名和 `timestamp`、`nonceStr`、`prepay_id`、`signType` 一起返回客户端。这个返回值就是客户端调用 `wx.requestPayment` 调起微信支付所需要的全部数据。

官方文档对以上流程描述的并不清楚，所以我们多花了一些笔墨讲解这个流程。代码清单 12-33 中的 `prepareSignData` 函数实现了上述签名和打包数据的功能。



在 `prepareSignData` 函数中,实例化了一个 `WxPayJsApiPay` 对象,这个对象是微信 SDK 中专门用来返回签名及签名数据的类。随后读取了 `appId`,生成了 `timeStamp` 时间戳、`nonceStr` 随机数,设置了签名方式和 `prepay_id`。然后调用 `WxPayJsApiPay` 的 `MakeSign` 方法生成了一个包括 `appId`、`timeStamp`、`nonceStr`、签名方式和 `prepay_id` 的签名。最后将这个签名连同以上被签名的值返回小程序中。

这里要特别注意 3 个地方。第一个地方是 `package` 一定不能直接等于 `prepay_id`,必须将其设置为 `'prepay_id'.'.orderNo['prepay_id']`,否则小程序调用 `wx.requestPayment` 时会报错;第二个地方是记得最后将 `appId` 从返回的明文数据数组中删除,小程序不需要 `appId` 就可以拉起微信支付;第三个地方是极有可能遇到一个错误提示“curl 出错, 错误码:60”。解决这个问题的方法是修改一下 `WxPay.Api.php` 中的代码。找到 `WxPay.Api.php` 文件的第 537 行:

代码清单 12-34 解决 curl 错误码 60 的问题

WxPay.Api.php

找到以下代码:

```
curl_setopt($ch,CURLOPT_SSL_VERIFYPEER,TRUE);
curl_setopt($ch,CURLOPT_SSL_VERIFYHOST,2);
```

将其修改为:

```
curl_setopt($ch,CURLOPT_SSL_VERIFYPEER,FALSE);
curl_setopt($ch,CURLOPT_SSL_VERIFYHOST,FALSE);
```

以上代码修改完成后,服务器将返回客户端小程序调用 `wx.requestPayment` 所需要的全部参数。

最后,我们来修改一下 Orange Can 项目中的 `wx-pay.js` 代码。给出 `wx-pay.js` 的完整代码如下:

代码清单 12-35 小程序最终拉起微信支付

wx-pay.js

```
Page({
  data: {},
  onTap: function () {
    wx.login({
      success: function (res) {
        console.log('code:' + res.code);
        wx.request({
          url: "http://127.0.0.1:8080/wxopen/wxpay.php",
          data: {
            code: res.code
          },
          success: function (res) {
            var preData = res.data;
            console.log(preData);
```



在微信支付界面完成支付行为并支付成功后，微信将收到一条支付成功的消息。

在本章之前所有示例中，我们反复调用 `wx.login` 获取 `code`，并将 `code` 发送到开发者服务器换取 `openId` 和 `session_key`。正如在前几个章节中反复强调的，在 `session_key` 的有效时间内（未过期），开发者应该自行维护 `session key` 而不是重复获取。

我们来看一下微信官方建议的登录以及登录状态维护指导流程，如图 12-18 所示。

开发者服务器在收到 code 码后，协同 appid、appsecret 以及 code 码向微信服务器换取当

前用户的 session key 和 openId。

获取到 `session_key` 和 `openId` 后不要将这两个变量发送回客户端。此时应自己生成一个令牌（随机字符串，也就是图中描述的 `3rd_session`）作为用户 `session_key` 和 `openId` 的键，将令牌（键）和 `session key`、`openId`（值）保存到服务器的 Redis 或者 Memcache 中。

同时, 这个自己生成的令牌应当具有一定时效性, 时效性一定要小于 30 天。开发者可根据自己小程序的安全性要求自行调整这个令牌的有效期。对于微信中的令牌字符串的要求有一些建议, 请查看图 12-18 中 3rd_session 的说明。

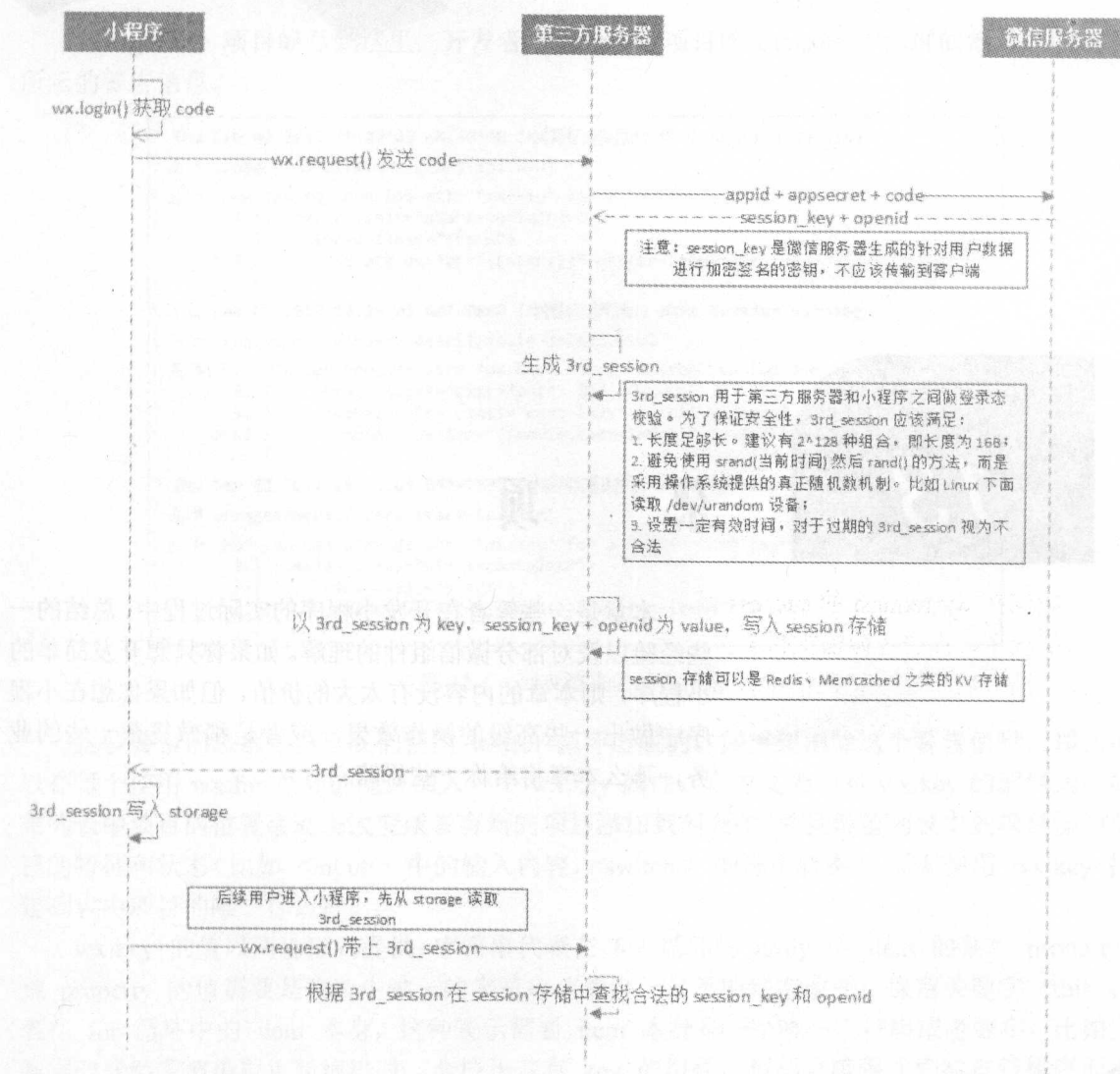


图 12-18 微信官方建议的小程序登录及登录状态维护流程图

开发者应当将自己生成的令牌发送到小程序并存入小程序的缓存中, 每次请求服务器时都携带这个令牌, 服务器接收令牌后在 Redis 中查找用户真实的 openid 和 session_key。

以上流程只是一个指导性的流程, 在真实的项目中还有很多变化和细节, 但在总体思路应当遵守上述流程。

第 13 章

杂 项

本章是一些笔者在开发小程序的实际过程中，总结的一些经验以及对部分微信组件的理解。如果你只想开发简单的小程序，则本章的内容没有太大的价值，但如果你想在小程序中做出一些高级的操作效果，或者是稍微复杂一些的业务，那么本章将给你一些思路。

13.1 wx:key

Orange Can 项目编写到这里，开发者可能会发现项目的 Console 中满屏的都是如图 13-1 所示的警告信息。

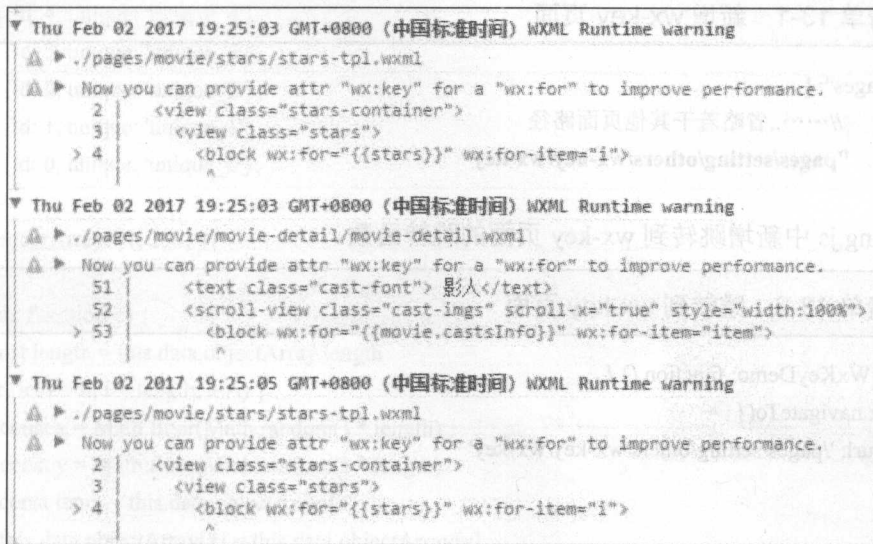


图 13-1 列表渲染的警告信息

这些警告信息都是由于我们使用 `wx:for` 循环造成的。如果想消除这个警告信息，那么可以在每个使用 `wx:for` 循环的地方加入一个 `wx:key` 属性。官方文档中对 `wx:key` 的解释为：如果列表中项目的位置会动态改变或者有新的项目添加到列表中，并且希望列表中的项目保持自己的特征和状态（比如 `<input/>` 中的输入内容，`<switch/>` 的选中状态），需要使用 `wx:key` 指定列表中项目的唯一标识符。

`wx:key` 的值以两种形式提供：字符串代表在 `for` 循环的 `array` 中 `item` 的某个 `property`，该 `property` 的值需要是列表中唯一的字符串或数字，且不能动态改变；保留关键字 `*this` 代表在 `for` 循环中的 `item` 本身，这种表示需要 `item` 本身是一个唯一字符串或者数字，比如当数据改变触发渲染层重新渲染时，会校正带有 `key` 的组件，框架会确保他们被重新排序而不是重新创建，以确保使组件保持自身的状态并提高列表渲染时的效率。

如果不提供 `wx:key`，会报一个警告信息。如果明确知道该列表是静态的或者不必关注其顺序，可以选择忽略。

官方的解释并不那么容易理解，简单来说，如果想消除这个警告信息，那么可以使用 `wx:key` 属性。`wx:key` 的取值有两种形式：如果 `for` 循环数组中的子项是一个唯一不重复的字符串或者数字，那么使用 `wx:key="*this"` 即可；如果 `for` 循环的子项是一个复杂的 `Object` 对象，那么可以使用这个 `Object` 对象下某一个属性名作为 `wx:key` 的取值，前提条件是这个属性是一个唯一的字符串或数字，且不能被动态改变。

以 Orange Can 项目中电影数据为例, 豆瓣电影数据的 id 号非常适合做 for 循环的 key 值。开发者可自行将豆瓣电影数据的 id 号作为 wx:key 的取值添加到每一个使用 wx:for 循环的标签上。

消除警告并不是最终目的, 还要理解 wx:key 的作用到底是什么。可以在 setting 页面下编写一个事例看看 wx:key 的作用。在 app.json 的 pages 数组下新增 wx-key 页面用来演示 wx:key 标签属性的使用方法。

代码清单 13-1 新增 wx-key 页面

app.json

```
"pages": [  
  //.....省略若干其他页面路径  
  "pages/setting/others/wx-key/wx-key"  
]
```

在 setting.js 中新增跳转到 wx-key 页面的跳转函数。

代码清单 13-2 跳转到 wx-key 页面

setting.js

```
showWxKeyDemo: function () {  
  wx.navigateTo({  
    url: '/pages/setting/others/wx-key/wx-key'  
  });  
}
```

下面编写一个小 demo, 在 wx-key.wxml 中增加以下代码:

代码清单 13-3 wx:key 演示代码

wx-key.wxml

```
<switch wx:for="{{objectArray}}" style="display: block;">  
  {{item.id}}  
</switch>  
<button bindtap="switch">  
  Switch  
</button>  
<button bindtap="addToFront">  
  Add to the front  
</button>  
<switch wx:for="{{numberArray}}" wx:key="*this" style="display: block;">  
  {{item}}  
</switch>  
<button bindtap="addNumberToFront">  
  Add to the front  
</button>
```

接着, 在 wx-key.js 文件中添加以下代码:



代码清单 13-4 wx:key 演示示例代码

wx-key.js

```

Page({
  data: {
    objectArray: [
      {id: 5, unique: 'unique_5'},
      {id: 4, unique: 'unique_4'},
      {id: 3, unique: 'unique_3'},
      {id: 2, unique: 'unique_2'},
      {id: 1, unique: 'unique_1'},
      {id: 0, unique: 'unique_0'},
    ],
    numberArray: [1, 2, 3, 4]
  },
  switch: function(e) {
    const length = this.data.objectArray.length
    for (let i = 0; i < length; ++i) {
      const x = Math.floor(Math.random() * length)
      const y = Math.floor(Math.random() * length)
      const temp = this.data.objectArray[x]
      this.data.objectArray[x] = this.data.objectArray[y]
      this.data.objectArray[y] = temp
    }
    this.setData({
      objectArray: this.data.objectArray
    })
  },
  addToFront: function(e) {
    const length = this.data.objectArray.length
    this.data.objectArray = [{id: length, unique: 'unique_' + length}].concat(this.data.objectArray)
    this.setData({
      objectArray: this.data.objectArray
    })
  },
  addNumberToFront: function(e){
    this.data.numberArray = [ this.data.numberArray.length + 1 ].concat(this.data.numberArray)
    this.setData({
      numberArray: this.data.numberArray
    })
  }
})

```

保存并运行代码，wx-key 页面呈现如图 13-2 所示的效果。



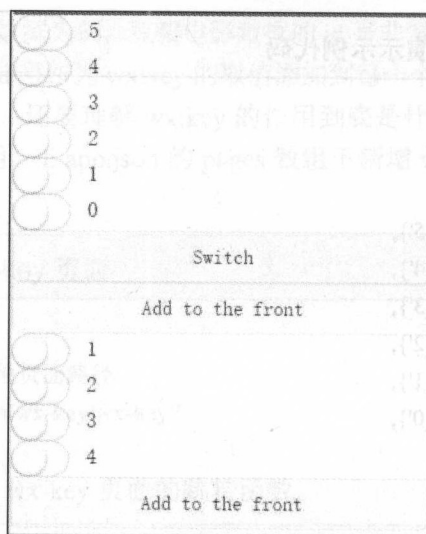


图 13-2 wx-key 页面效果图

Wx-key 页面默认有两组 switch 组件和 3 个 button。首先来解释 Switch 按钮的功能。点击 Switch 按钮后将触发 switch 事件响应函数，switch 函数以随机方式重新排列 objectArray，然后执行数据绑定，更新被打乱顺序的数组。这将使页面 UI 上的 6 个元素随机重新排列。而 Add to the front 按钮将新增一个 switch 开关，并置于 objectArray 数组的首部，这将不断在页面 UI 的 switch 组的“顶部”新增 switch 开关。

我们来看一个“奇怪”的现象。随机将 0~5 号 switch 组件的其中一个开关“打开”，比如将 3 号开关打开。打开后 3 号开关将呈现绿色的样式。此时不断点击 Add to the front 按钮或者 switch 按钮，会发现绿色开关始终处于固定不变的位置，但开关后面的数字在不断变化。这显然是不合理的，我们打开 3 号开关，3 号开关的状态被设置为“打开”，但页面被 setData 更新后，3 号开关的状态不再是“打开”了，这样的结果在真实项目中将造成很大的“bug”。

我们的目的是打开 3 号开关，那么无论页面如何变动（渲染），3 号开关始终应当保持被打开的状态，而其他开关应该保持未打开的状态，这样的业务逻辑才是合理的。

那么如何解决这个问题呢？

这时就是 wx:key 发挥作用的时候了。当数据改变触发渲染层重新渲染时，会校正带有 key 的组件，框架会确保他们被重新排序而不是重新创建，以确保组件保持自身的状态并提高列表渲染时的效率。

下面来看具体的效果。在第一个 switch 组件中添加一个 wx:key="unique" 属性。注意 unique 是 objectArray 子对象的其中一个属性，我们将这个属性的属性名称作为 wx:key 的键。保存代码并再次运行项目，此时再打开一个开关，并不断点击 Switch 或者 Add to the front 按钮，发现打开的开关会不断“跳动”，且被打开的开关和开关后面的数字始终是对应的。比如打开 3 号开关，无论如何重排、添加，3 号开关始终会保持打开的状态。

所以，对于一些带有状态的组件（比如 input 和 switch），再做列表渲染时应当考虑加入 wx:key。

示例中的第二个 switch 组件列表与第一个 switch 组件列表所展示的功能基本一样，只不过它描述了 wx:key 取值的另一种方法，使用 *this 为 wx:key 提供键。因为第二个 switch 组件的数据来源是一组数字而不是一组 object 对象，对于数字和字符串类型的数据，可以使用 *this。

13.2 scroll-view 组件：在 js 中控制滚动条

在电影详情页面中，我们介绍了 scroll-view 的基本用法。scroll-view 组件的用法很多，除了可以用来实现一些复杂的滚动效果外，也最能体现如何使用数据绑定的方法模拟操作 dom 的效果。

先来看 scroll-view 的全部属性和事件列表：

- scroll-x Boolean 类型，默认值为 false，允许横向滚动。
- scroll-y Boolean 类型，默认值为 false，允许纵向滚动。
- upper-threshold Number 类型，默认值为 50，距顶部/左边多远时（单位 px）触发 scrolltoupper 事件。
- lower-threshold Number 类型，默认值为 50，距底部/右边多远时（单位 px），触发 scrolltolower 事件。
- scroll-top Number 类型，设置竖向滚动条位置。
- scroll-left Number 类型，设置横向滚动条位置。
- scroll-into-view String 类型，值为某子元素 id 就滚动到该元素，元素顶部对齐滚动区域顶部。
- bindscrolltoupper 事件，滚动到顶部/左边时会触发 scrolltoupper 事件。
- bindscrolltolower 事件，滚动到底部/右边时会触发 scrolltolower 事件。
- bindscroll 事件，滚动时触发，可以从事件的事件参数中获取滚动的位置以及滚动偏移量。具体参数格式为：event.detail = {scrollLeft, scrollTop, scrollHeight, scrollWidth, deltaX, deltaY}

scroll-view 的基本功能是提供一个可控制横向、纵向滚动条的容器，基本使用方法非常简单。scroll-view 最主要的功能是当滚动条滚到到底部或右边、上部或左边时，会触发 bindscrolltoupper 和 bindscrolltolower 事件。开发者可在这两个事件中编写自己的业务逻辑。比如上滑加载更多数据就可以使用 scroll-view 组件，当 scroll-view 组件滚动到底部时，在 bindscrolltolower 事件响应函数中编写加载更多数据的业务代码，并重新绑定数据即可实现加载更多数据的功能。如果想控制触发 bindscrolltoupper 和 bindscrolltolower 事件的“敏感度”，就可以使用 upper-threshold 和 lower-threshold 两个属性。这两个属性可以让开发者指定距离底部/右边、顶部/左边还有多远时触发 bindscrolltoupper 和 bindscrolltolower 事件。

除此之外，scroll-view 还提供了一个 bindscroll 事件。只要 scroll-view 组件被滚动，就会触发这个事件。如果想实现一个 scroll-view 组件和另一个组件联动的效果，那么可以考虑整个事件。例如，左边是商品分类列表，右边是分类子项，当滚动右边的分类子项时，左边的分类列表会一起联动。

以上几个属性都比较好理解，这一节主要演示 scroll-top、scroll-left 和 scroll-into-view 这 3 个属性的用法和效果。

在 dom 结构的 Web 网页中，使用 JavaScript 操作滚动条是非常简单的，但在小程序中没有办法用“获取一个组件，再操作组件”的思想编程。再次强调，我们只有数据绑定这一种方式，一定要“忘掉”传统网页里经典的 DOM 编程方式。

那么如何实现使用 js 控制滚动条的位置？小程序提供的 scroll-view 组件有 3 个属性用于在 js 中控制滚动条滚动到指定的位置：croll-top、scroll-left 和 scroll-into-view。

下面编写一个小示例，看看如何在 js 中控制 scroll-view 的滚动条。在 app.json 的 page 数组下新增一个 scroll-view 页面。

代码清单 13-5 新增 scroll-view 页面

app.json

```
"pages": [
  //.....省略若干其他页面
  "pages/setting/others/scroll-view/scroll-view"
]
```

接着，在 setting.js 中新增点击【scroll-view 高级用法演示】按钮跳转到 scroll-view 页面的跳转方法。

代码清单 13-6 跳转到 scroll-view 页面

setting.js

```
showScrollViewDemo: function () {
  wx.navigateTo({
    url: '/pages/setting/others/scroll-view/scroll-view'
  });
}
```

在 scroll-view.wxml 中新增以下代码：

代码清单 13-7 scroll-view 的骨架代码

scroll-view.wxml

```
<view>
  <view>vertical scroll</view>
  <scroll-view class="scroll_view_V" scroll-y="true" style="height: 200px;"
    bindscrolltoupper="upper" bindscrolltolower="lower" bindscroll="scroll"
    scroll-into-view="{{toView}}" scroll-top="{{scrollTop}}">
    <view id="green" class="item-size green"></view>
    <view id="red" class="item-size red"></view>
    <view id="yellow" class="item-size yellow"></view>
    <view id="blue" class="item-size blue"></view>
    <view id="SlateGray" class="item-size SlateGray"></view>
    <view id="GoldEnrod" class="item-size GoldEnrod"></view>
  </scroll-view>
```



```

<view class="btn-area">
  <button size="mini" bindtap="tap">click me to scroll into view </button>
  <button size="mini" bindtap="tapMove">click me to scroll</button>
</view>
</view>
<view >
  <view >horizontal scroll</view>
  <scroll-view class="scroll_view_H" scroll-x="true" style="width: 100%">
    <view id="green" class="item-size green item"></view>
    <view id="red" class="item-size red item"></view>
    <view id="yellow" class="item-size yellow item"></view>
    <view id="blue" class="item-size blue item"></view>
    <view id="SlateGray" class="item-size SlateGray item"></view>
    <view id="GoldEnrod" class="item-size GoldEnrod item"></view>
  </scroll-view>
</view>

```

下面给出 scroll-view.wxss 的样式代码。

代码清单 13-8 scroll-view 页面的样式代码

scroll-view.wxss

```

.item-size {
  height: 70px;
  width: 70px;
}

.red {
  background-color: red;
}

.yellow {
  background-color: yellow;
}

.blue {
  background-color: blue;
}

.SlateGray {
  background-color: SlateGray;
}

```

```

.GoldEnrod{
  background-color:GoldEnrod;
}

.green{
  background-color:green;
}

.scroll_view_H{
  white-space: nowrap;
}

.scroll_view_V{
  display:flex;
  flex-direction: column;
}

.item{
  display:inline-block;

```

在 scroll-view 页面中添加了两组 scroll-view 组件，一组垂直方向，一组水平方向。当然，这里并不是为了演示 scroll-view 组件的基本用法，重点要看看 scroll-into-view、scroll-top 两个属性的作用。

在 scroll-view.js 中新增以下代码：

代码清单 13-9 scroll-view 页面的逻辑代码

scroll-view.js

```

var order = ['red', 'yellow', 'blue', 'green', 'red',
'SlateGray', 'GoldEnrod']

Page({
  data: {
    toView: 'green',
    scrollTop: 0
  },
  upper: function(e) {
    console.log(e)
  },
  lower: function(e) {
    console.log(e)
  },
  scroll: function(e) {
    console.log(e)
  },

```




```
tap: function(e) {
  //不断更新 scroll-into-view 属性的取值
```

```
  for (var i = 0; i < order.length; ++i) {
```

```
    if (order[i] === this.data.toView) {
```

```
      this.setData({
        toView: order[i + 1]
```

```
      })
```

```
      break
```

```
    }
```

```
  },
```

```
  tapMove: function(e) {
```

```
    //不断更新 scroll-top 的取值
```

```
    this.setData({
```

```
      scrollTop: this.data.scrollTop + 10
```

```
    })
```

```
  }
```

```
})
```

编写完以上 js 代码后，scroll-view 页面将呈现如图 13-3 所示的效果。

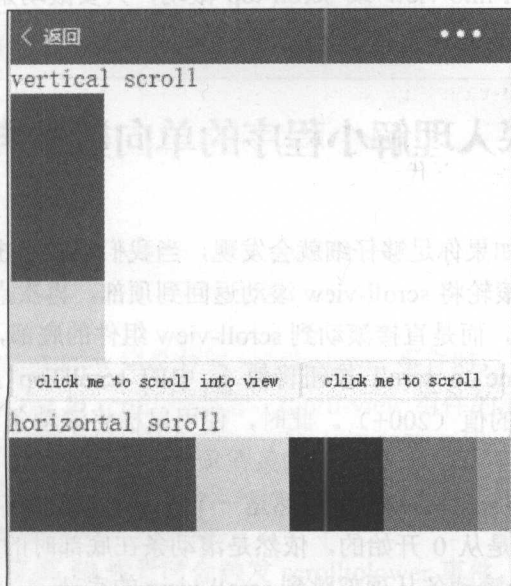


图 13-3 scroll-view 页面的 UI 效果

可以看到，一组 scroll-view 是垂直方向的，一组 scroll-view 是水平方向的，这两组 scroll-view 下都有若干子元素。控制 scroll-view 组件滚动条方向的属性是 scroll-x 和 scroll-y，垂直方向的 scroll-view 一定要设置一个高度，滚动条将在高度限定的范围内滚动。

注意页面中有两个按钮：click me to scroll into view 和 click me to scroll，这两个按钮将控制垂直方向 scroll-view 的滚动情况，开发者可以先点击这两个按钮看一下效果。

每次点击左边的按钮，垂直 `scroll-view` 将滑动一个方块的距离；每次点击右边的按钮，垂直 `scroll-view` 下的子元素将移动 10。这就是 `scroll-into-view` 和 `scroll-top` 的效果。

`scroll-into-view` 指定一个 `scroll-view` 内某子元素的 id，被指定的子元素的顶部将对齐滚动区域的顶部。所以当我们在 js 中不断变换 `scroll-into-view` 的取值时，就会出现“一格一格跳动”的效果。每变换一次，`scroll-into-view` 就会被指定一个新的子元素 id，同时这个子元素将被移动到 `scroll-view` 的顶部。

`scroll-top` 指定的是垂直滚动条的位置，每次点击 `click me to scroll` 按钮都会将 `scrollTop` 的值增加 10，这样滚动条的位置就会向下移动 10，`scroll-view` 的子元素也将一起向下移动 10。

如果开发者足够细心会发现一个有趣的现象，如果一直点击 `click me to scroll into view` 按钮，子元素会“循环”滚动，因为 `scroll-into-view` 只和其指定的元素 id 有关系。

但是如果一直点击 `click me to scroll`，就不会出现“循环”效果。因为 js 中的 `scrollTop` 会被一直增加，甚至超过 `scroll-view` 的高度(200+)。超过高度后，即使 `scrollTop` 被累加到 1000+，`scroll-view` 也不可能再往下滚动了。

除此之外，当我们不断点击 `click me to scroll into view` 和 `click me to scroll` 两个按钮时，`upper`、`lower` 和 `scroll` 这 3 个事件响应函数也会不断执行，并向 Console 面板中不断输出信息。请开发者注意观察 Console 面板。`upper` 是当页面滚动到顶部时触发的事件响应函数；`lower` 是当页面滚动到底部时触发的事件响应函数；`scroll` 当页面滚动时就会触发，无论我们用鼠标滚轮滚动，还是使用 `scroll-into-view` 或 `scroll-top` 滚动，只要滚动条动了，`scroll` 函数就会被触发。

13.3 深入理解小程序的单向数据绑定机制

在上一节的示例中，如果你足够仔细就会发现，当我们不断点击 `click me to scroll` 按钮让滚动条触底后，再用鼠标滚轮将 `scroll-view` 滚动返回到顶部，再次点击 `click me to scroll` 按钮滚动条并不会向下移动 10，而是直接滚动到 `scroll-view` 组件的底部，这是为什么呢？

因为不断点击 `click me to scroll` 按钮将使 js 中的 `scrollTop` 累加到一个等于或者超出 `scroll-view` 组件最大高度的值(200+)。此时，使用鼠标将滚动条调回顶部将改变组件中的 `scroll-top` 属性值，让其恢复为 0 (0 代表滚动条在顶部)。但请注意，js 中的 `scrollTop` 变量并不会恢复到 0，此时 js 中的 `scrollTop` 变量依然是一个较大的数值，当再次点击 `click me to scroll` 按钮时，`scrollTop` 由于不是从 0 开始的，依然是滚动条在底部时的取值(200+)，因此当其再次累加 10 后，将直接让滚动条从顶部跳到 `scroll-view` 的底部。

从这个现象开发者应该理解了什么是小程序的单向数据绑定。可以通过控制 js 变量实现更新页面 UI (比如更新 `scrollTop` 变量将使滚动条不断滚动)，但却不能通过鼠标拖动滚动条更新 js 中的 `scrollTop` 变量。一个方向的更新是可以的，但反过来却不可以，这就是单向数据绑定。逻辑层上的数据变更可以反映到 UI 层上，但 UI 层上的数据变更却不会自动反馈到 js 逻辑层中。

再举个例子，回顾一下之前多次使用的 input 组件。我们经常在 input 组件的 value 属性上绑定一个 js 变量，当操作 js 变量的取值时，input 组件的输入内容会被 js 变量改变。如果反过来呢？用户在 input 组件中输入内容时 js 中被绑定的变量会自动根据内容的变更而改变吗？并不会。像这样的情况在小程序中比比皆是，因为小程序只实现了单向数据绑定。这和著名的 MVVM 框架 AngularJS 不同，AngularJS 可以双向数据绑定，当我们更改了 UI 上的某个属性值时，JS 中被绑定的变量也会同样被更新。

当然，双向数据绑定也有缺点，就是性能较差。所以没有绝对优秀的解决方案，每个不同的解决方案有其优势也同时会存在缺点。寄生在微信中的小程序本身性能就无法比拟原生 App，所以为了保证性能，暂时不支持双向数据绑定也可以理解。

13.4 深入理解 scroll-view 组件的 bindscrolltolower、lower-threshold 属性

下面我们对 scroll-view 页面的代码做一些变更，变更的目的是更好地理解 bindscrolltolower 事件和 lower-threshold 属性。

在 scroll-view.js 的 lower 函数中增加以下代码：

代码清单 13-10 修改 lower 函数

scroll-view.js

```
lower: function(e) {
  console.log(e)
  this.setData({
    scrollTop:0
  })
}
```

修改代码后，每当 scroll-view 组件触发 scrolltolower 事件时，都会将 scrollTop 重置为 0。此时，我们不断点击 click me to scroll 按钮时，scroll-view 组件的滚动条将可以实现“循环”滚动。因为当滚动条滚动到距底部一定距离时（这个距离是 50 像素，后面我们会解释为什么是 50 像素）将触发 scrolltolower 事件，这将导致 scrollTop 变量重置为 0，下一次点击 click me to scroll 按钮时，scrollTop 将从 0 开始累加。

下面解释为什么距离底部 50 像素时会触发 scrolltolower 事件。在 scroll-view 组件中有一个 lower-threshold 属性，当距底部/右边一定距离时（单位为 px）会触发 scrolltolower 事件，这个属性的默认值是 50 像素。我们可以修改这个属性，将其指定为 0，这样滚动条只有滚动到底部才会触发 scrolltolower 事件。

水平方向的 scroll-view 属性和事件的用法几乎和垂直方向的 scroll-view 组件一模一样，这里就不再赘述。但要注意水平方向 scroll-view 的 CSS 样式，关于水平方向 scroll-view 的注意事项我们已经在电影详情页面中详细讲解过。

scroll-view 的这些高级属性对于实现一些复杂效果非常有帮助，如果你想实现类似以下两种 App 中常见的功能效果，可以考虑使用 scroll-view 组件。其中，一个示例来自于 App “饿了么”，另一个示例来自于小程序“猫眼电影”，如图 13-4 和图 13-5 所示。

控制页面滑动是以上两张图中所实现效果的难点。scroll-view 组件可以帮助你实现这种效果。



图 13-4 “饿了么” App 选择食物页面

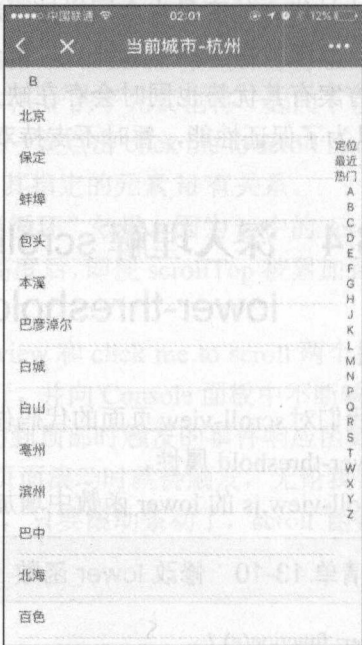


图 13-5 “猫眼电影”小程序中的城市选择

13.5 微信小程序发布流程

本来准备详细描述一下小程序的整体发布流程，但看了一下官方文档，微信官方已经给出了非常详细的从注册到开发再到发布的流程，这里就不再浪费纸墨，请开发者参考文档：<https://mp.weixin.qq.com/debug/wxadoc/introduction/#产品定位及功能介绍>。

作者简介



雷磊，8年研发及团队管理经验。曾就职于国内最大的GIS公司互联网部门，任职研发经理。精通C#、Python、Java、JavaScript等语言与Web开发技术。工作之余经常撰写有关互联网技术、商业模式等文章，发表在各个TMT媒体上。微信小程序首批内测开发者，知乎专栏“小楼昨夜又秋风”作者。

本书特色



本书主要围绕Orange Can项目展开一系列编码工作，用几近真实的项目介绍小程序的各个API、组件用法，小程序开发经验、技巧以及常见的误区说明。

整个Orange Can项目分为三部分：文章阅读、电影资讯及页面设置。文章阅读部分包括文章列表、文章详情及评论。通过编写文章阅读功能的代码，你将学会swiper组件的裁剪模式、image组件的裁剪模式、缓存的使用技巧、列表渲染、数据绑定、模板、音乐播放、录音、分享等知识。除此之外，你将对小程序页面的生命周期有一个大致的了解。学习完这部分内容，你将可以轻松地做出一个内容型小程序应用。

电影资讯部分将有助于读者学习如何调用服务器数据，掌握嵌套template（模板）的使用技巧及它的优势。

页面设置部分包含大量功能示例，包括获取硬件设备信息、罗盘与重力感应的应用、扫描二维码、用户登录、用户信息校验、解析用户加密数据、获取用户openid、发送模板消息、微信支付等功能。

本书不仅全面、深入地介绍了小程序开发中的各项技能，解答了编者在实际开发中踩到的“坑”，而且提供小程序开发中常见问题的解决方案，可以大大减少开发者的试错时间，是一本注重实践与解决问题的小程序开发首选工具书。

清华大学出版社数字出版网站

WQBook 书文局泉

www.wqbook.com

ISBN 978-7-302-46801-1



9 787302 468011 >

定价：69.00元